НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ім. ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Макашин М.С.

Завідувач кафедри

_____ Олександр Коваль

« __»_____2020 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: Рекомендаційна система з використанням методів машинного навчання для надання персоналізованих рекомендацій в реальному часі

Виконав:
студент IV курсу, групи ТМ-61
Макашин Михайло Сергійович _____

Керівник:
Професор кафедри АПЕПС, доктор технічних наук,
Отрох Сергій Іванович _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

‘Національний технічний університет України
“ Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)
«____» _____ 2020р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Макашину Михайлу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Рекомендаційна система з використанням методів машинного навчання для надання персоналізованих рекомендацій в реальному часі
Науковий керівник Отрох Сергій Іванович, д.т.н., професор
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “25” травня 2020року №___

2. Строк подання студентом роботи 12.06.2020

3. Вихідні дані до роботи: Персональний комп’ютер під керуванням операційної системи MacOS, хмарне сховище AWS, мови програмування Python, Javascript, середовище розробки Jupyter

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити аналіз існуючих алгоритмів систем для надання рекомендацій, вибір методів та інструментів для розробки колаборативної рекомендаційної системи та рекомендаційної системи на основі вмісту, розробка архітектури системи, вибір та реалізація засобів розробки програмного забезпечення, розробка веб-додатку з інтеграцією системи

5. Орієнтований перелік ілюстративного матеріалу Низькорозмірна факторизація матриць, схема роботи колаборативної системи, комплексна архітектура програмного продукту, гібридна система рекомендацій, діаграма прецедентів системи

6. Дата видачі завдання «__11__»____жовтня_____2019р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1	Затвердження теми роботи	03.02.2020	
2	Вивчення та аналіз задачі	13-19.04.2020	
3	Розробка архітектури та загальної структури системи	20-26.04.2020	
4	Розробка структур окремих підсистем	27.04-03.05.2020	
5	Програмна реалізація системи	04-13.05.2020	
6	Оформлення пояснювальної записки	14-16.05.2020	
7	Захист програмного продукту	17.05.2020	
8	Передзахист	9.06.2020	
9	Захист	17.06.2020	

Студент . _____ ____Макашин М.С._____
(підпис) (прізвище та ініціали)

Науковий керівник _____ ____Отрох С.І._____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мета роботи — аналіз існуючих методів машинного навчання і рекомендаційних систем, розробка нової рекомендаційної системи, яка буде поєднувати у собі колаборативну рекомендаційну систему та на основі вмісту, розробка веб-додатку, який буде мати зручний інтерфейс та демонструватиме роботу рекомендаційної системи.

Для вирішення поставленої задачі було сформовано перелік завдань, які визначили структуру дослідження:

— проаналізувати предметну область, визначити інструменти для програмної реалізації
— спроектувати модель для надання колаборативних рекомендацій та на основі вмісту
— визначення фільму, який з найбільшої вірогідністю сподобається користувачу на основі попередніх прийнятих рішень іншими користувачами
— визначення подібності фільмів за його описом
— можливість надання персоналізованих рекомендацій користувачу
— розробка веб-додатку з інтеграцією системи

Для розв’язання поставлених задач, використовувались мови програмування Python, Javascript, алгоритми машинного навчання, а саме:

— чергування найменших квадратів для реалізації колаборативної рекомендаційної системи
— низькорозмірна факторизація матриць
— нейронні мережі для репрезентації слів у вектор

Записка містить 70 сторінок, 22 рисунки, 2 додатки і 12 посилань.

Ключові слова: машинне навчання, нейронні мережі, чергування найменших квадратів, низькорозмірна факторизація матриць, колаборативна рекомендаційна система, система рекомендацій на основі вмісту.

ABSTRACT

The purpose of the work is to analyze existing machine learning methods and recommendation systems, develop a new recommendation system that will combine a collaborative recommendation system and content-based recommendation system, develop a web application that will have a user-friendly interface and demonstrate the recommendation system.

For the tasks set, the roll-out was formed, the designation of the structure was as follows:

— analyze the subject area, tools for implementation

— design a model for adding collaborative recommendations based on the user experience

— determining the movie that the user is most likely to like based on previous decisions made by other users.

— determining the similarity of films according to his description

— the ability to provide personalized recommendations to the user

— development of a web-based system with an integrated system

To solve the tasks, Python programming languages, Javascript, machine learning algorithms were used:

— alternative least square

— low-dimensional matrix factorization

— neural networks for the representation of words in a vector

The note contains 70 pages, 22 figures, 2 appendices and 12 links.

Keywords: machine learning, neural networks, least squares alternation, low-dimensional matrix factorization, collaborative recommendation system, content-based recommendation system.

ЗМІСТ

ВСТУП

З кожним днем програмне забезпечення впливає на наше життя все більше і більше. Виникають нові проблеми, які слід вирішувати шляхом автоматизації. Одним з найбільш перспективніших напрямів сьогодні є машинне навчання. Ще в 1959 році Артур Семюель запровадив цей термін, але широкого розповсюдження це набуло відносно нещодавно.

Всім відомо, що інформація править світом. Великі корпорації, за роки свого існування зібрали тисячі терабайтів даних, але вони не приносять ніякої користі якщо їх не обробляти. Зрозуміло, що одна людина, чи навіть сотні людей, за все своє життя не зможуть опрацювати навіть і частини цієї інформації, а її кількість збільшується з кожним днем. У нагоді стає машинне навчання, вже створено сотні алгоритмів, бібліотек, фреймворків, що дозволяють застосовувати цю технологію для власних досліджень. Це дає змогу будувати трьох-вимірні зображення органів людини, що допомагає хірургам краще підготуватись до операцій, розпізнавати обличчя, що збільшує шанси знаходження злочинців, а також складати персоналізовані рекомендації, на основі вподобань користувача.

Сфера розваг в інтернеті набуває більшої популярності. COVID-19 став каталізатором сервісів, які надають онлайн послуги. Оскільки конкуренція - це двигун прогресу, інженери щодня працюють над новими алгоритмами, рішеннями, що можуть покращити їх продукт. В результаті цього виникло таке поняття як рекомендаційна система.

Рекомендаційна система - це підклас системи фільтрації інформації, яка прагне передбачити “рейтинг” або “уподобання”, який користувач надав би предмету. Оскільки ця проблема ще не була вирішена до кінця, було запропоновано дослідити можливість застосування алгоритмів машинного навчання для надання користувачу рекомендацій в реальному часі.

1. ПОСТАНОВКА ЗАДАЧІ НАДАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ В РЕАЛЬНОМУ ЧАСІ

За останнє десятиріччя рекомендаційні системи набули широкого поширення в інтернет-магазинах, соціальних мережах, сервісах для перегляду кінофільмів. Вони дозволяють не тільки утримати наявних користувачів, а й залучити нових.

Важливою проблемою, яку вирішують рекомендаційні системи, є показ контенту, який з максимальною вірогідністю зацікавить користувача. В таких випадках потрібно проаналізувати, що раніше сподобалось користувачу, які елементи є максимально подібними з даним, чи переглядав користувач даний контент раніше.

Подібні системи використовуються в різноманітних областях і найчастіше зустрічаються в сервісах для перегляду відео-контенту (Netflix, Youtube) , інтернет-магазинах (Amazon), соціальних мережах (Facebook, Instagram).

Тому було запропоновано дослідити алгоритми машинного навчання для побудови рекомендаційної системи і її використання для вирішення даної проблеми.

Як результат зробленого дослідження, потрібно створити веб-додаток, основними функціями якого є:

— Можливість визначення фільму, який з найбільшої вірогідністю сподобається користувачу на основі попередніх прийнятих рішень іншими користувачами.
— Можливість визначення подібності фільмів за його описом.
— Можливість надання персоналізованих рекомендацій користувачу.
— Забезпечити можливість реєстрації та авторизації користувачів.

— Забезпечити користувача особистим кабінетом, в якому можна додавати вподобані фільми та керувати персональними даними.
— Забезпечити можливість оцінки фільму користувачем.
— Забезпечити можливість збереження фільму.
— Забезпечити інтерфейс адміністратора для створення та редагування фільмів, акторів, жанрів.
— Забезпечити користувача зручним інтерфейсом.

Висновки до розділу 1

Задача надання персоналізованих рекомендацій полягає в визначенні наступного елемента, який ймовірно сподобається користувачу, на основі попередніх рішень інших користувачів та подібності інформації, яку містить даний елемент.

2. АНАЛІЗ ІСНУЮЧИХ МЕТООДІВ НАДАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

Особливістю задачі надання персоналізованих рекомендацій є отримання джерела набору даних та методів машинного навчання завдяки яким буде проводиться тренування інтелектуальної системи. Розглянемо вже існуючі методи та системи надання рекомендацій.

2.1. Колаборативна рекомендаційна система

Колаборативна рекомендаційна система – це найбільш затребувана, широко впроваджена та найбільш досліджена технологія, яка доступна ринку. Колаборативні системи рекомендацій об’єднують рейтинги об’єктів, розпізнають схожість вподобань користувачів на основі їх попередніх оцінок та генерують рекомендації, порівнюючи їх. Дана система ґрунтується на припущенні, що люди, які мали схожі інтереси в минулому, будуть мати їх і в майбутньому.

2.2. Система рекомендацій на основі вмісту

У даній системі об’єкти в основному визначаються пов’язаними з ними ознаками. Рекомендаційна система на основі вмісту вивчає профіль інтересів нового користувача на основі наявних функцій в об’єктах, які користувач оцінив.

Це в основному система рекомендацій для ключових слів, тут ключові слова використовуються для опису елементів. Таким чином, дана система працює за таким алгоритмом, що вона надає подібні елементи користувачу до тих, які сподобались йому у минулому.

2.3. Демографічна система рекомендацій

Демографічна система рекомендацій спрямована на групуванні користувачів на основі демографічних даних. Багато галузей використовували такий підхід, оскільки він не такий складний у реалізації. У системі рекомендацій, що базується на демографічних даних, алгоритми спочатку потребують належного дослідження ринку у вказаному регіоні, а також короткого опитування для збору даних для кластеризації користувачів. Перевага даного методу полягає в тому, що він не потребує даних з попередніми оцінками користувачами об’єктів, як у колаборативній та на основі вмісту рекомендаційних системах.

2.4. Рекомендаційна система на основі знань

Рекомендаційна система на основі знань намагається запропонувати об’єкти на основі висновків про потреби та вподобання користувача. Дана система працює на основі функціональної інформації: вона має дані про те, як певний предмет задовільняє потребу користувача і тому може міркувати про зв’язок між потребою та можливою рекомендацією.

2.5. Гібридна система рекомендацій

Гібридна система рекомендацій – це поєднання будь-яких двох систем таким чином, що одна з них відповідає конкретній галузі. Це найбільш затребувана система рекомендацій, яку досліджують багато компаній, оскільки вона поєднує сильні сторони двох або більше систем для надання рекомендацій, а також усуває слабкі місця, які існують, коли використовується лише одна система. Є кілька способів комбінування систем:

— Обчислення результатів усіх доступних рекомендаційних систем. Наприклад, P-Tango комбінує колаборативну та на основі вмісту рекомендаційні системи, дає їм рівнозначну вагомість на початку роботи програми, але постійно коригує значення, оскільки передбачення щодо об’єктів, які сподобаються користувачам підтверджуються або ні.

— Переключенні між двома система в залежності від ситуації. Даний спосіб може використовувати спочатку одну систему для надання рекомендацій, але якщо передбачення не справджуються, то він може переключитися на іншу.

Висновки до розділу 2

Таким чином, було проаналізовано п’ять існуючих методів для надання персоналізованих рекомендацій. В результаті аналізу можна виділити наступне:

— Існуючі рекомендаційні системи, такі як P-Tango, має реалізовані функції для надання рекомендацій, але вони пристосовані тільки для специфічної

області, а також наражають дані користувачів ресурсу на небезпеку, оскільки їх дані будуть оброблятись на сторонньому сервісі.

— моделі у існуючих фреймворках, були натреновані на конкретному наборі даних, який характерний для користувачів їхнього ресурсу, це робить систему не універсальною для інтеграції.

3. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Аналізуючи поставлену задачу, було прийнято рішення, розробити веб-додаток, оскільки цей спосіб не потребує встановлення додаткових програм для запуску продукту і є універсальним як і для мобільних пристроїв, так і для персональних комп’ютерів. Веб-додаток розташований на веб-сервісі і не потребує ресурсів пристроїв користувача для обробки даних, вибраний спосіб значно пришвидшує швидкість роботи програми.

3.1. Вибір архітектури програмного комплексу

Для реалізації поставленої задачі була розроблена комплексна архітектура, яка складається з декількох компонентів: сервер для обробки користувацьких дій на веб-ресурсі та надання персоналізованих рекомендацій, база даних, клієнт і сервер для взаємодії з всіма компонентами. Схема даної архітектури зображена на рисунку 3.1.

Головним у програмному комплексі є сервер 1, він виступає в ролі “моста” між іншими компонентами системи, об’єднувавши їх. Він також містить у собі основну бізнес логіку та логіку доступу до даних. Надає можливість для аутентифікації та авторизації користувача. Сервер 1 приймає HTTP запити з клієнтського компоненту, оброблює їх, записуючи зміни до бази даних, якщо потрібно, або ж надсилає інформацію для обробки до серверу 2.

Сервер 2 відповідає за обробку користувацьких дій у веб-додатку, та створенням персоналізованих рекомендацій. Він отримує HTTP запит з даними з серверу 1, аналізує отриману інформацію та віддає відповідь з інформацією, який контент потрібно показати користувачу.

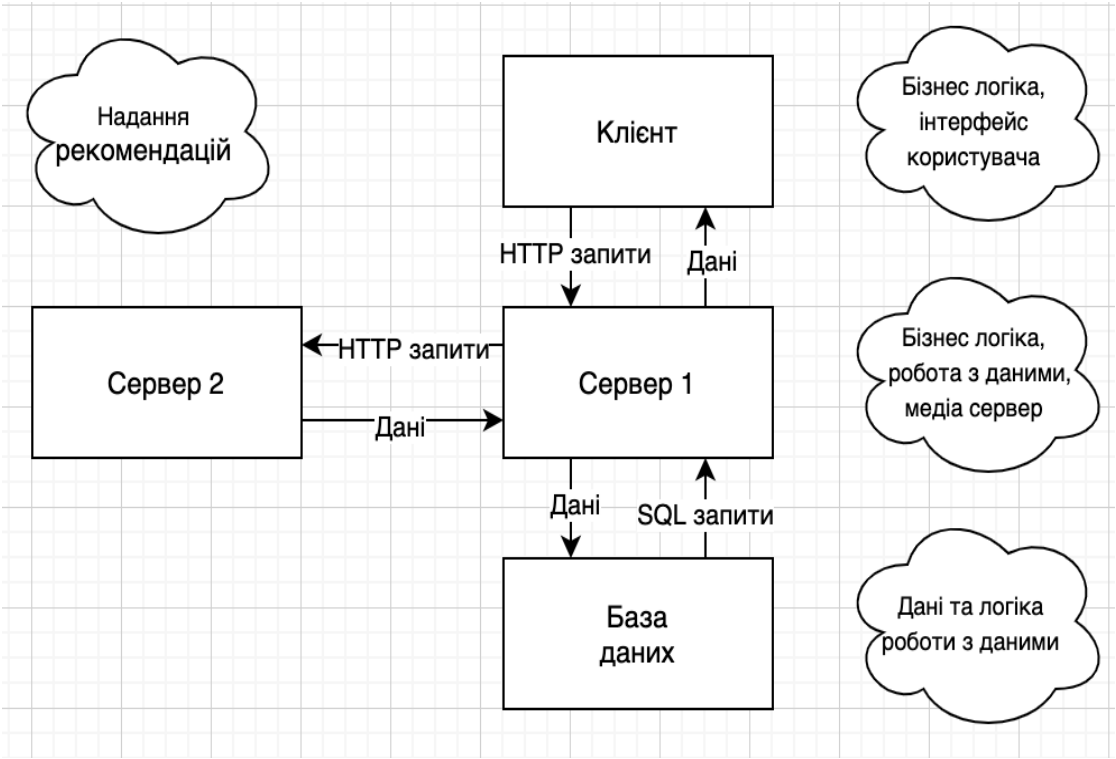

Рисунок 3.1 – Комплексна архітектура програмного комплексу

Користувач не може спостерігати, що відбувається на серверній частині додатку. Він співпрацює з клієнтським компонентом, на ньому відбувається вся взаємодія з користувачем. Клієнтський компонент реагує на дії користувача, попередньо обробляє дані та відправляє запит на сервер. Він отримує вказівки про подальші дії і виводить результат отриманий з серверу 1.

Важливою задачею рівня бази даних є забезпечення збереження даних, які сервер зберігає для подальшого використання. Також забезпечується цілісність даних за допомогою зовнішніх зв’язків та ключів.

3.2. Опис архітектури серверу 1 і 2

Шаблон проектування — це ефективне рішення, яке описує спосіб вирішення задачі проєктування програмного забезпечення.

Для реалізації серверу було використано фреймворк, який є реалізацією шаблону проектування MVT .

MVT (Model-View-Template) — це шаблон проектування, головною ідеєю якого є відділити логіку застосунку від представлення (рисунок 3.2). Принципом MVT є розділення програмної реалізації системи на три головні компоненти: M — Model (Модель), V — View (Представлення), T — Template (Шаблон), таким чином, що редагування будь-якого компонента може відбуватися незалежно.

Модель — рівень доступу до даних. До неї відноситься все, що пов’язано з даними - як отримати до них доступ, як перевірити їх, яка у них поведінка та зв’язки один з одним.

Представлення — це міст між моделями та шаблонами. Включає в себе всю логіку для здійснення зв’язку між моделями, обробки даних та передачі їх до шаблону.

Шаблон - це рівень який відповідає за відображення даних на сторінці, та взаємодію з користувачем.

Основна мета застосування цієї концепції полягає в відділенні бізнес-логіки (моделі) від її візуалізації (представлення) [1].

3.3. Опис архітектури клієнтського застосунку

Для реалізації клієнтського компоненту системи, було вибрано фреймворк, в основу якого було покладено шаблон проектування MVVM (рисунок 3.3).

Model-View-ViewModel (MVVM) — архітектурний шаблон, орієнтований головним чином на платформи, що підтримують зв’язування даних та елементів користувацького інтерфейсу. MVVP переслідує ті ж цілі, що і MVT. Розділити бізнес-логіку і користувацький інтерфейс. Однак в MVT зміни, зроблені користувачем при роботі з інтерфейсом, не впливають безпосередньо на модель, а попередньо обробляються контролером. В MVVM використовується двостороннє зв’язування.

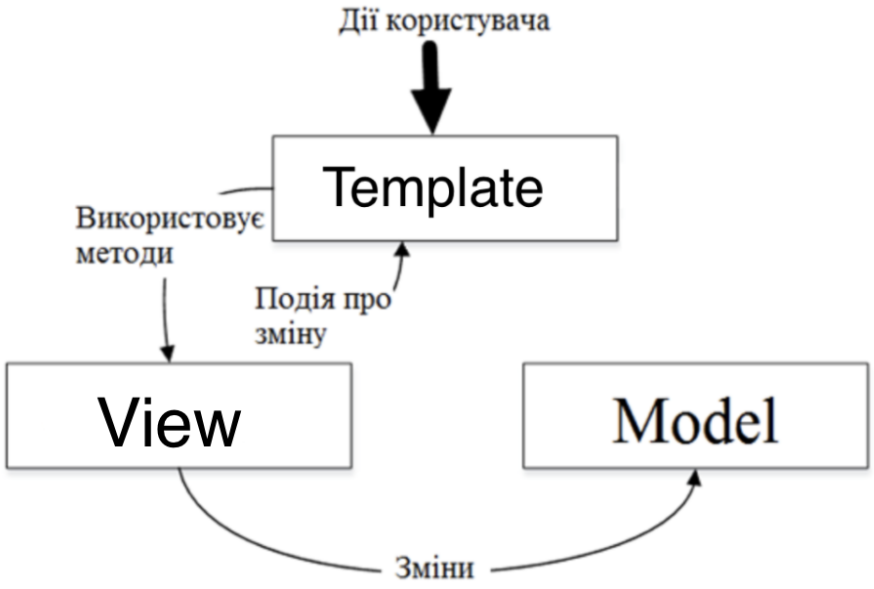

Рисунок 3.2 – Схема роботи MVT шаблону

Шаблон MVVM ділиться на три частини:

— Модель (Model) - являє собою бізнес логіку застосунку.

— Представлення (View) - графічний інтерфейс для роботи з даними та їх відображення.

— Модель вигляду (ViewModel, що означає “Model of View”) - обгортка підлягаючих скріпленню даних з моделі, яка містить методи, що використовуються представленням для роботи з самою моделлю.

Рисунок 3.3 – Схема роботи MVVM шаблону

3.4. Опис інструментів розробки

Оскільки веб-застосунок складається з декількох частин, для різних компонентів системи були вибрані найбільш зручні інструменти.

Для клієнтського компоненту було використано мову програмування JavaScript з бібліотекою Backbone, мови розмітки HTML/SCSS та фреймворк для створення графічних інтерфейсів Foundation.

Backbone — це JavaScript бібліотека з RESTful JSON інтерфейсом і базується на парадигмі програмування Model-View-Presenter (MVP). Backbone.js відомий своїми малими розмірами та прямою залежністю від бібліотеки Underscore.js. Бібліотека призначена для розробки односторінкових веб-застосунків. Backbone.js відомий своїми малими розмірами та прямою залежністю від бібліотеки Underscore.js. Бібліотека призначена для розробки односторінкових веб-застосунків. Розроблена Джеремі Ашкінсоном, відомим завдяки CoffeeScript. [2]

Foundation — це набір інструментів з відкритим кодом, призначений для створення веб-сайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків.[3]

Для серверного рівня було обрано такі технології: мова програмування Python та фреймворк Django, Django ORM для роботи з базою даних.

Python - мова програмування високого рівня, орієнтована на підвищення продуктивності, розробки та легкості читання коду. Синтаксис ядра Python мінімалістичний. У той час, коли стандартна бібліотека включає великий об'єм корисних функцій. Python підтримує структурне, об'єктно-орієнтоване, функціональне, імперативне та аспектно-орієнтоване програмування. Основні архітектурні чорти - динамічна типізація, автоматичне управління пам’яттю, механізм обробки помилок, підтримка багатопоточних обчислень, високорівневих структур даних. [4]

Django - це веб-система Python високого рівня, яка дозволяє швидко розробити безпечні та бездоганні веб-сайти. Побудований досвідченими розробниками, Django піклується про багато клопоту веб-розробки, тому ви можете зосередитись на написанні програми, не потребуючи винаходити колесо. Це безкоштовний та відкритий код, має процвітаючу та активну спільноту, чудову документацію та безліч варіантів безкоштовної та платної підтримки.

Для рівня бази даних було обрано PostgreSQL.

PostgreSQL — об'єктно-реляційна система керування базами даних. Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення. PostgreSQL використовувався для реалізації великих систем, таких як: система аналізу фінансових даних, пакет моніторингу функціональності потоків, база даних відстеження астероїдів, система медичної інформації, кілька географічних систем. POSTGRES також використовувався як навчальний інструмент в кількох університетах. 1992 року POSTGRES став головною СКБД наукового комп'ютерного проекту Sequoia 2000 [5].

На другому сервісі для проведення розрахунків та для побудови моделі було використано наступні технології:

Pandas — програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами.

Numpy — розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами. Numpy можна розглядати як гарну вільну альтернативу MATLAB, оскільки мова програмування MATLAB зовні нагадує Numpy: обидві вони інтерпретовані, і обидві дозволяють користувачам писати швидкі програми поки більшість операцій проводяться над масивами або матрицями, а не над скалярами. [6]

Apache Spark - це платформа паралельної обробки з відкритим кодом, яка підтримує обробку в пам'яті, щоб підвищити продуктивність додатків, які аналізують великі дані. Рішення для роботи з великими даними призначені для обробки даних із занадто великим обсягом або складністю для традиційних баз даних. Spark обробляє великі обсяги даних в пам'яті, що набагато швидше, ніж альтернативна обробка з використанням диска. Spark може використовуватися як у типових сценаріях оброблення даних, схожих на MapReduce, так і для реалізації специфічних методів, таких як потокове оброблення, SQL, інтерактивні та аналітичні запити, рішення задач машинного навчання і робота з графами. Програми для оброблення даних можуть створюватися на мовах Scala, Java, Python та R. [7]

Kubernetes - це система для автоматичного розгортання, масштабування та управління застосунками у контейнерах. Kubernetes необхідний для безперервної інтеграції і поставки програмного забезпечення (CI / CD, Continuos Integration / Continuos Delivery), що відповідає DevOps-підходу. Завдяки

«упаковці» програмного оточення в контейнер, мікросервіс можна дуже швидко розгорнути на робочому сервері (production), безпечно взаємодіючи з іншими додатками.[8]

3.5. Особливості вирішення задачі надання персоналізованих рекомендацій за допомогою колаборативного фільтрування

Ідея колаборативного фільтрування полягає в тому, що маючи великий набір вподобань користувачів до певних об’єктів, можна передбачити можливі вподобання, які наразі відсутні (рисунок 3.4).

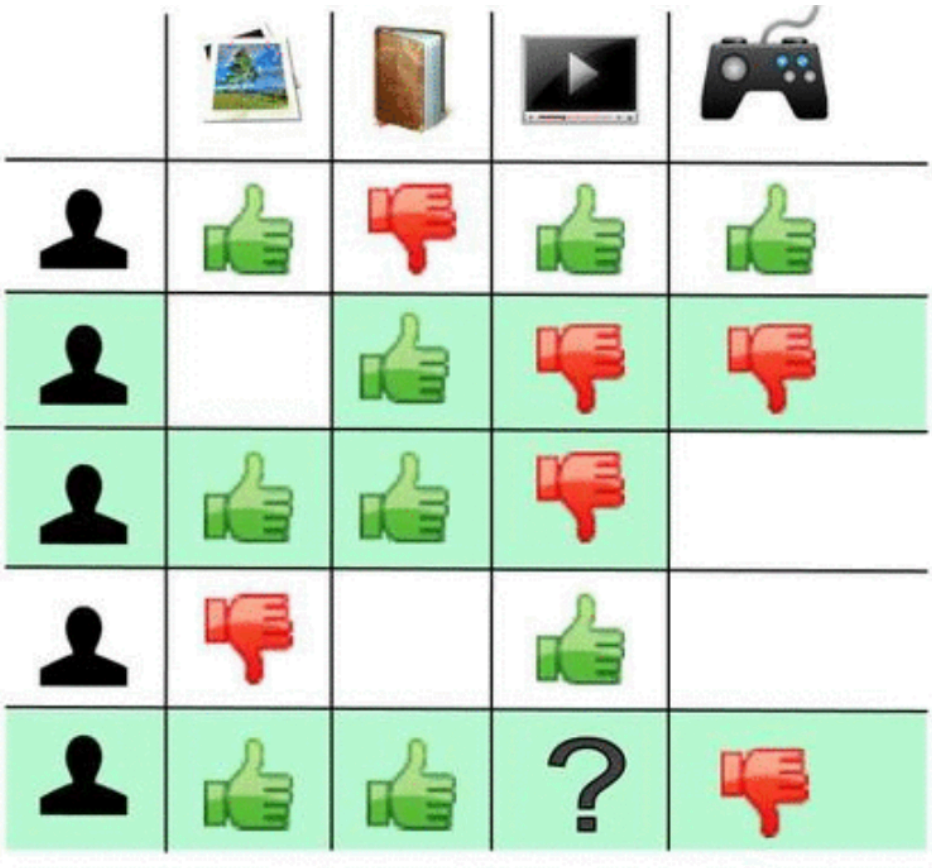

Рисунок 3.4 – Візуалізація роботи колаборативної системи.

Найпоширенішим методом для здійснення цього є низькорозмірна факторизація матриць за допомогою методу найменших квадратів.

Головна ідея будь-якого методу факторизації матриці – це знайти менший набір даних прихованих факторів, які можна використовувати для прогнозування відсутніх записів. Оскільки не всі користувачі оцінили кожен з фільмів, ми не знаємо всіх записів цієї матриці, саме тому нам потрібна колаборативна фільтрація. Для кожного користувача наявні рейтинги лише для підмножини фільмів. Ідея полягає в апроксимації матриці оцінок, розподіливши її як добуток двох матриць: однієї, яка описує властивості кожного користувача (показана зеленим кольором), і тієї, що описує властивості кожного фільму (показано синім кольором) (рисунок 3.5.).

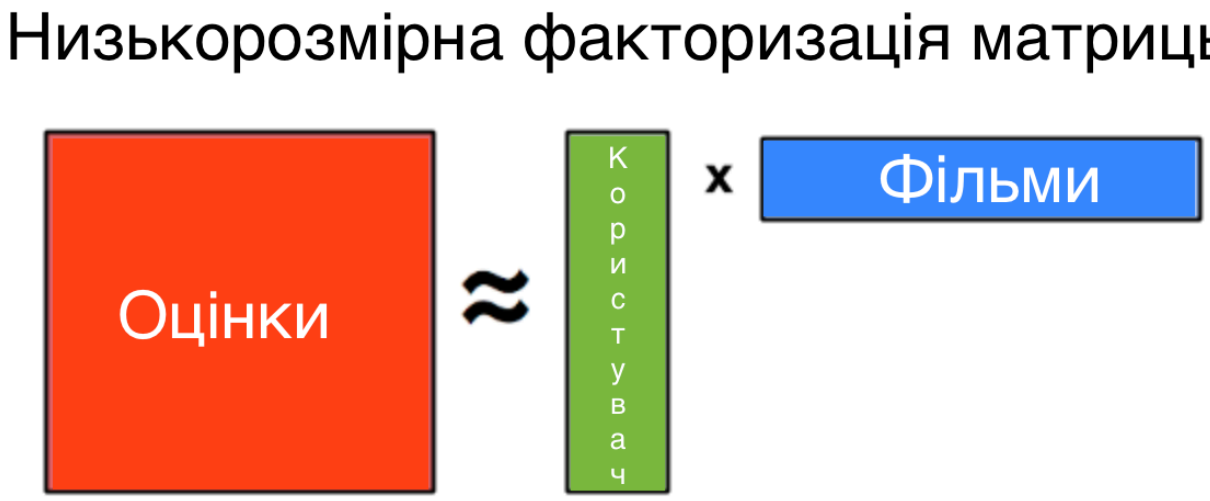

Рисунок 3.5 – Низькорозмірна факторизація матриць

Потрібно визначити ці дві матриці таким чином, щоб похибка для пар користувач/фільм, де ми знаємо правильний рейтинг, була мінімальною. Метод найменших квадратів робить це, заповнюючи спочатку матрицю фільмів випадковими значеннями, а потім оптимізовує їх для зменшення похибки. Потім він використовує матрицю фільмів як константу і оптимізовує значення матриці

користувача.

Другим методом, що використовується в колаборативній фільтрації є сингулярне розкладання. Даний метод декомпозує матрицю на 3 інших матриці, що обчислюються за формулою 3.1:

$$A = USV^T \qquad (3.1)$$

Де A матриця $m \times n$, U ортогональна матриця $m \times n$, S діагональна матриця $n \times n$, S ортогональна матриця $n \times n$.

Приховані фактори – це характеристики предметів, наприклад, жанр фільму. Сингулярне розкладання зменшує розмірність матриці корисності A шляхом вилучення її прихованих факторів. Воно відображає кожного користувача та кожного елемента в r-мірному прихованому просторі. Таке відображення показує чітке представлення відносин між користувачами та предметами. Нехай кожен елемент представлений вектором x_i, а кожен користувач представлений вектором y_u. Очікувана оцінка користувачем елементу може бути дана як γ_{ui}, що обчислюється за формулою 3.2:

$$\gamma_{ui} = x_i^T y_u \qquad (3.2)$$

У даному випадку γ_{ui} – це форма факторизації в сингулярному розкладанні.

x_i і y_i можна отримати таким чином, що різниця квадратних похибок між певним елементом та очікуваною оцінкою в матриці елемента користувача є мінімальною. Це можна виразити як показано на формулі 3.3:

$$Min(x, y) \sum_{(u,i)\in K} (r_{ui} - x_i^T y_u)^2 \qquad (3.3)$$

Для зменшення похибки між значенням, передбаченим моделлю, і фактичним значенням, алгоритм використовує метод зміщення. Нехай для пари елементів користувача (u, i), μ - середня оцінка всіх елементів, b_i середній рейтинг елементу $i - u$ і b_u – це середній рейтинг наданий користувачем $u - \mu$, остаточне рівняння після додавання регуляризації та зміщення може бути наведене у вигляді формули 3.4:

$$Min(x, y, b_i,\ b_u) \sum_{(u,i)\in K}(r_{ui} - x_i^T y_u - \mu - b_i - b_u)^2 + \lambda(\left|\left|x_i\right|\right|^2 + \left|\left|y_u\right|\right|^2 + + b_i^2 + b_u^2) \quad (3.4)$$

Користувачі і елементи можуть бути представлені в вигляді точок в k-вимірному просторі. Представимо користувача вектором з k факторів r_u і кожен продукт вектором з k факторів r_i, щоб передбачити рейтинг користувача u товару *i*, обчислюєм їх скалярний добуток за формулою 3.5:

$$r_{u,i} = r_i\, r_u = r_i^T r_u \quad (3.5)$$

Можна сказати, що вектор факторів користувача показує, наскільки користувачу подобається чи не подобається той чи інший фактор, а вектор факторів фільму показує, наскільки той чи інший фактор в продукті виражений.

3.6. Особливості вирішення задачі надання персоналізованих рекомендацій за допомогою визначення схожості вмісту

Існують різні типи векторів: двохвимірні, трьохвимірні чи n-вимірні. Для двохвимірних векторів скалярний добуток між двома ідентичними векторами дорівнює їхньому модулю в квадраті, тоді як вони перпендикулярні, скалярний добуток дорівнює нулю. Як правило, для n-вимірних векторів скалярний добуток можна обчислити за формулою 3.6.

$$uv = [u_1\ u_2 \dots u_n] \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^{n} u_i v_i \quad (3.6)$$

Скалярний добуток є важливим для визначення подібності, так як він безпосередньо пов’язаний з цим. Визначення подібності між двома векторами *u* та *v* є, власне, співвідношення між їх скалярним добутком та добутком їхніх величин, що можна обчислити за формулою 3.7.

$$similarity = \cos(\theta) = \frac{uv}{\|u\|\|v\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}} \quad (3.7)$$

Значення подібності буде рівне 1, якщо два вектори ідентичні, і 0, якщо вони ортогональні. Іншими словами, подібність це число, що належить проміжку від 0 до 1, яке показує, наскільки два вектори схожі.

3.7. Особливості вирішення задачі перетворення тексту у вектор

Не так давно компанія Google розробила метод Word2Vec, який враховує контекст, і в той же час зменшує об'єм даних.

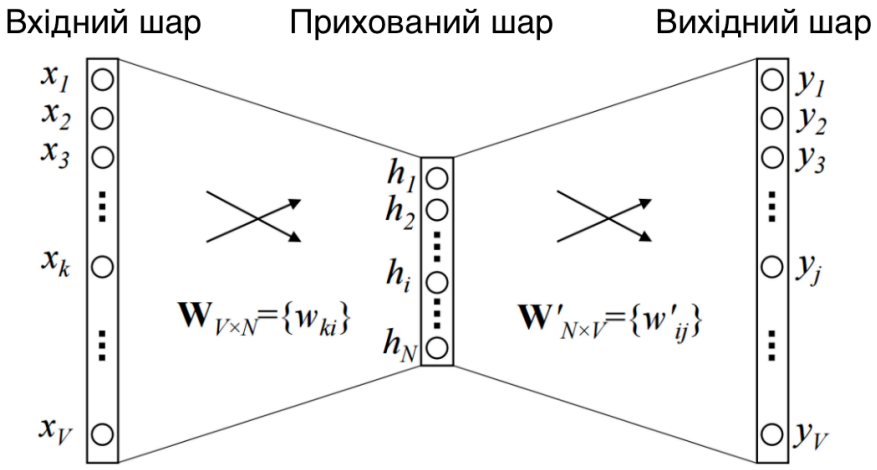

Рисунок 3.6. – Демонстрація роботи неперервного мішку слів.

Насправді Word2Vector представляє два різних методи: неперервний мішок слів та skip-gram. Задачою першого методу є передбачення слова на основі слів, що знаходяться поблизу, у skip-gram зворотня задача.

Вхідне або контекстне слово – це один зашифрований вектор розміром V.

Прихований шар містить N нейронів, а вихідний шар знову ж вектор розміром V. На рисунку 3.6. $W_{V\times N}$ – вагова матриця, яка відображає вхід x у прихований шар ($V \times N$ розмірна матриця). $W'_{N\times V}$ – вагова матриця, яка відображає приховані виходи шару на кінцевий вихідний шар ($N \times V$ розмірна матриця). Нейрони прихованого шару копіюють зважену суму входів у наступний шар.

Описана вище модель використовувала лише одне контекстне слово для прогнозування. Можна використовувати кілька контекстних слів (Рисунок 3.7.)

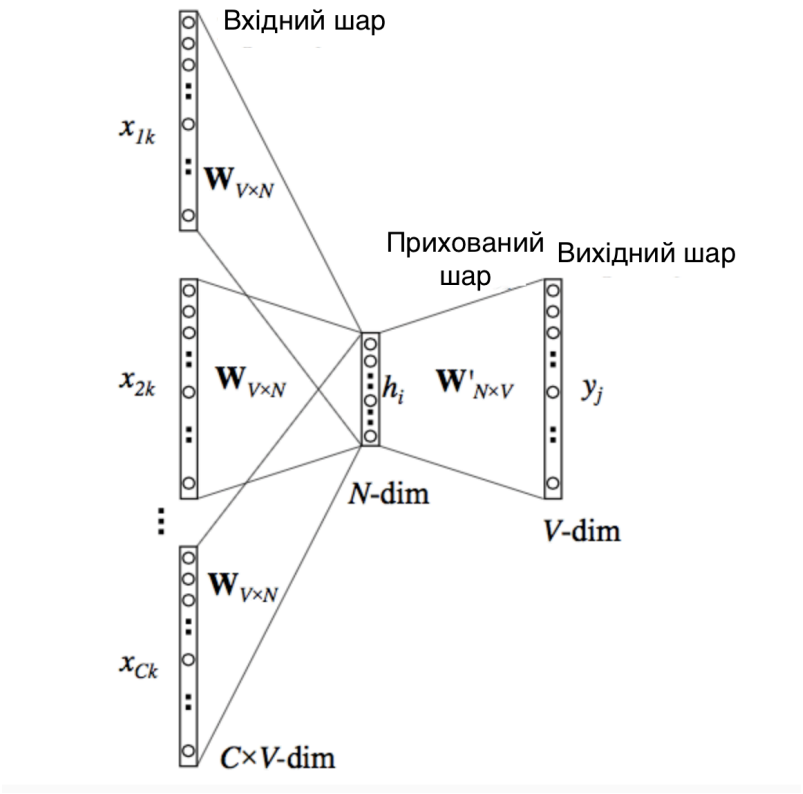

Рисунок 3.7. – Демонстрація роботи неперервного мішку слів з використанням декількох контекстних слів.

Проблему векторизації тексту також можна вирішити за допомогою TF-IDF методу. TF-IDF – це статистичний показник, що використовується для оцінки важливості слів у контексті документа, що є частиною колекції документів чи корпусу. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання слова у інших

документах колекції.[9]

Він має багато застосувань, основний метод в автоматизованому аналізі тексту і є дуже корисним для оцінки слів в алгоритмах машинного навчання для обробки природніх мов (NLP). TF-IDF був винайдений для пошуку документів та пошуку інформації. Він працює, збільшуючись пропорційно до кількості разів появи певного слова в документі, але компенсується кількістю документів, які містять слово. Якщо певне слова з'являється в документі багато разів, а в інших не з'являється велику кількість разів, то це, ймовірно, означає, що воно дуже актуальне. Приклад пошуку слів в документі можна побачити на рисунку 3.8.[10]

Terms	Docs
Book	Doc1 , Doc3 , Doc4
Teach	Doc4 , Doc61 , Doc103
.	.
.	.
.	.

Рисунок 3.8. Приклад пошуку слів в документі.

TF-IDF для слова в документі обчислюється множенням двох різних показників:

— частота слова в документі;

— зворотна документна частота слова в наборі документів. Це означає наскільки поширеним є слово у всьому наборі документів.

Множення цих двох показників призводить до оцінки TF-IDF слова у документі. Чим вище оцінка, тим релевантнішим є слово у цьому конкретному документі; Для більш формального математичного вираження, оцінка TF-IDF для слова t у документі d з набору D обчислюється за формулою 3.8:

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D) \qquad (3.8)$$

Де $tf(t,d)$ обчислюється за формулою 3.9, а $idf(t,D)$ за формулою 3.10:

$$tf(t,d) = log\big(1 + freq(t,d)\big) \qquad (3.9)$$

$$idf(t,D) = \log\left(\frac{N}{count(d \in D : t \in d)}\right) \qquad (3.10)$$

3.8. Комбінування колаборативної рекомендаційної системи та рекомендаційної системи на основі вмісту

З огляду на кілька моделей машинного навчання, які вміло вирішують проблему, але різними способами, існує проблема у комбінації результатів. Рішенням цього питання є використання іншої моделі машинного навчання.

Архітектура моделі комбінування включає дві або більше базові моделі, які часто називають моделями нульового рівня, і мета-моделі, що поєднує прогнози базових моделей. Моделі нульового рівня вже надали певні рекомендації, а мета-модель навчається на даних рекомендаціях для того, щоб краще поєднувати прогнози базових моделей. (Рисунок 3.9)

Найбільш розповсюдженим підходом до підготовки даних для мета-моделі полягає в перехресній валідації базових моделей. Дані для тренування мета-моделі також можуть включати початкові дані.

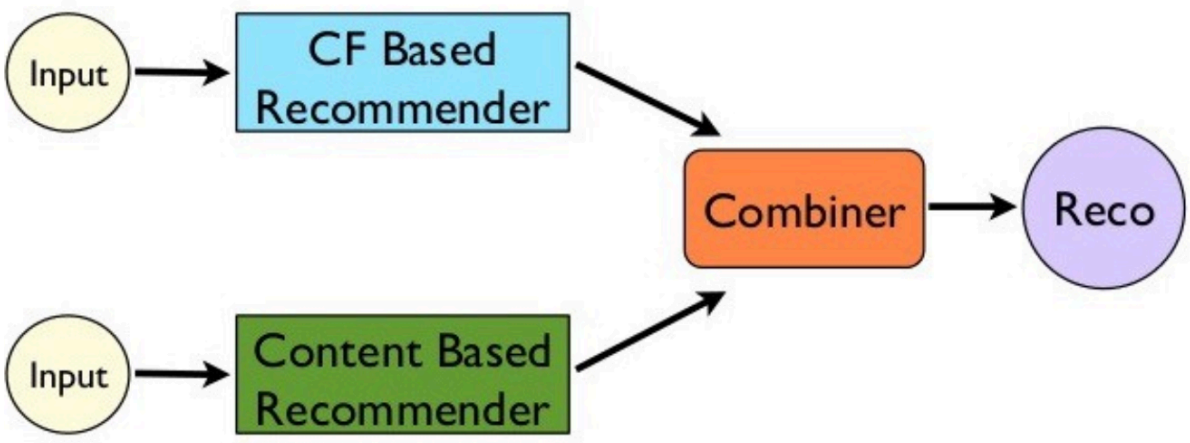

Рисунок 3.9. — Гібридна система рекомендацій

Мета-модель забезпечує чітку інтерпретацію прогнозів, зроблених базовими моделями. Таким чином, для тренування мета-моделі часто використовують лінійну регресію для регресійних завдань (прогнозування числового значення) та логічну регресію для завдань класифікації (прогнозування мітки класу).

3.9. Обґрунтування вибору програмної реалізації

Проектуючи систему, було прийнято до уваги такі фактори: швидкість роботи, масштабованість, зручність. Тому було прийнято рішення розробити веб-застосунок, адже це дозволяє обробляти дані на стороні веб-сервера, а не на стороні клієнта, не потребує додаткового програмного забезпечення для запуску, а також працює не залежно від операційної системи.

В якості основної мови програмування обрано Python. Python є інтерпретованою мовою програмування високого рівня загального призначення. Створений Гвідо ван Россумом і вперше випущений в 1991 році, Python має

філософію дизайну, яка підкреслює читабельність коду, особливо використовуючи значні пробіли [11].

Фреймворк Django дає змогу створювати швидкі та надійні веб-сервери будь-якої складності. Не потрібно вигадувати велосипед, адже даний засіб розробки містить усі необхідні механізми безпеки та підтримку HTTP запитів. Django ORM, що входить до даного фреймворку, забезпечує можливість працювати з базою даних працюючи з Python кодом, замість того, щоб створювати SQL-запити і надсилати їх до бази даних на пряму. Django також містить адмін-панель, що дозволяє керувати даними у веб-додатку (рисунок 3.9).

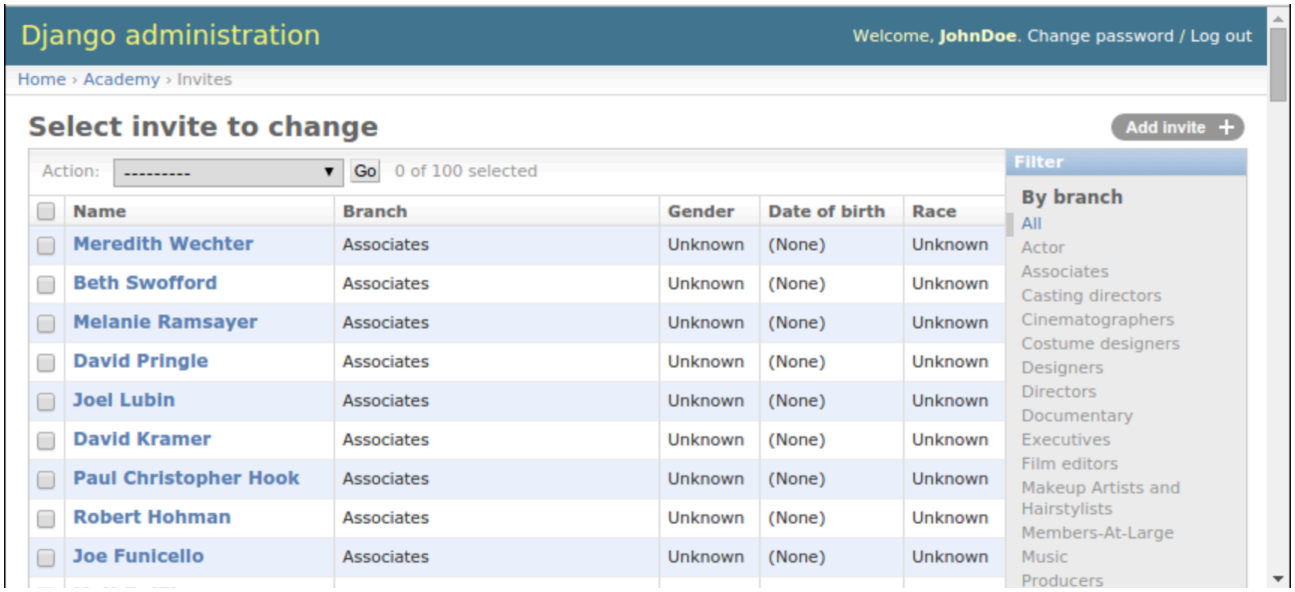

Рисунок 3.10. — Адмін-панель Django

На клієнтській частині було обрано Backbone.js, це не нова технологія, але вона було довго популярною із-за своєї надійності. Вона допомагає будувати швидкі користувацькі інтерфейси і реалізовувати частинну бізнес логіку на клієнтській частині.

Pandas і NumPy дають змогу швидко опрацьовувати математичні операції та візуалізовувати дані. Для того, щоб зчитати дані з файлу чи розрахувати кореляцію, ці бібліотеки є чудовим застосунком.

Spark - нова технологія, яка набула своєї популярності в роботі з великими даними. Оскільки для тренування моделі був використаний набір даних з 6 мільйонів записів, зберігати дані в реляційній базі даних не зручно, оскільки на проведення операції з ними витрачалося б багато часу. У даному випадку інформація зберігається в CSV файлі у хмарному сховищі. Spark SQL дозволяє працювати з записами в CSV файлі, як з звичайною реляційною базою даних, але робить це значно швидше.

В Spark існує концепція RDD (стійкий розподілений набір даних) — незмінна розподілена колекція об’єктів, які можна обробляти паралельно. В RDD можуть зберігатись об’єкти будь-яких типів.

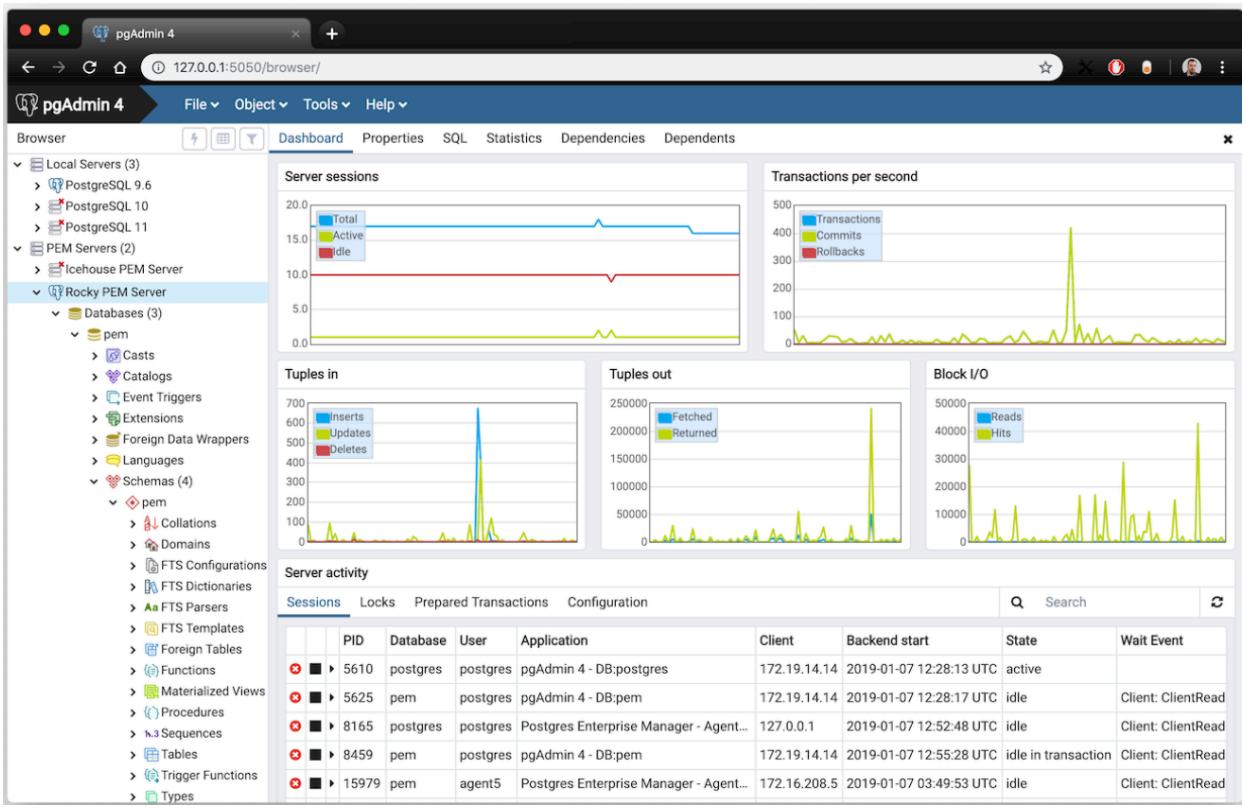

Рисунок 3.11. — pgAdmin

MLlib — це бібліотека для машинного навчання, що надає різноманітні алгоритми, що розроблені для горизонтального масштабування в кластері в цілях класифікації, регресії, кластеризації. Деякі з цих алгоритмів працюють з потоковими даними — наприклад, лінійна регресія з використанням звичайного методу найменших квадратів.[12]

Для тренування моделі колаборативної фільтрації було використано алгоритм Alternative Least Square, який показав найвищу точність серед інших алгоритмів.

В якості алгоритму конвертації слів у вектор було обрано алгоритм word2vec, оскільки отримані вектори зберігають семантичні і синтаксичні патерни.

Базою даних було обрано PostgreSQL, адже це найсучасніша реляційна база в наш час, адже до неї постійно додають нові можливості. Одними з найсильніших сторін даної бази даних є повнотекстовий пошук та кількість можливих індексів, яка є значно більшою ніж в конкурента MySQL. Для адміністрування бази даних було використано графічне відображення pgAdmin (рисунок 3.10).

Перечислені вище технології дають змогу збудувати швидкий та надійний продукт. Кожна з них - це застосунок з відкритим кодом, що є дуже важливим, тому є змога змінювати їх функціонал та бути захищеним від нових релізів творця фреймворків, адже інколи вони можуть зруйнувати створений на основі них застосунок.

Висновки до розділу 3

Таким чином, в результаті аналізу предметної області, а також вимог, які були висунуті при проектуванні, було обрано та обґрунтовано засоби реалізації системи надання персоналізованих рекомендацій, проведено ряд експерементів та тренувань моделей різноманітними методами, для визначення найефективніших методик.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Система передбачування вподобань користувача складається з декількох модулів, деякі з них мають функціональні підблоки. Головним модулем є форма для оцінки фільмів, які користувач буде бачити після того як увійде в систему. Це вирішить проблему “холодного пуску”, адже це новий користувач і необхідно мати базову інформацію про нього для того, щоб робити передбачення. Дані будуть надходити до серверу, де інформація буде оброблятись, зберігатись до бази даних та відправлятись до іншого серверу для складання передбачень. В результаті буде отримано набір фільмів з рейтингом від 1 до 5, де 5 - фільм найбільш вірогідно сподобається користувачу, а 1 - найменш.

На рисунку 4.1 наведена схема структури системи, на якій розташовані всі програмні модулі.

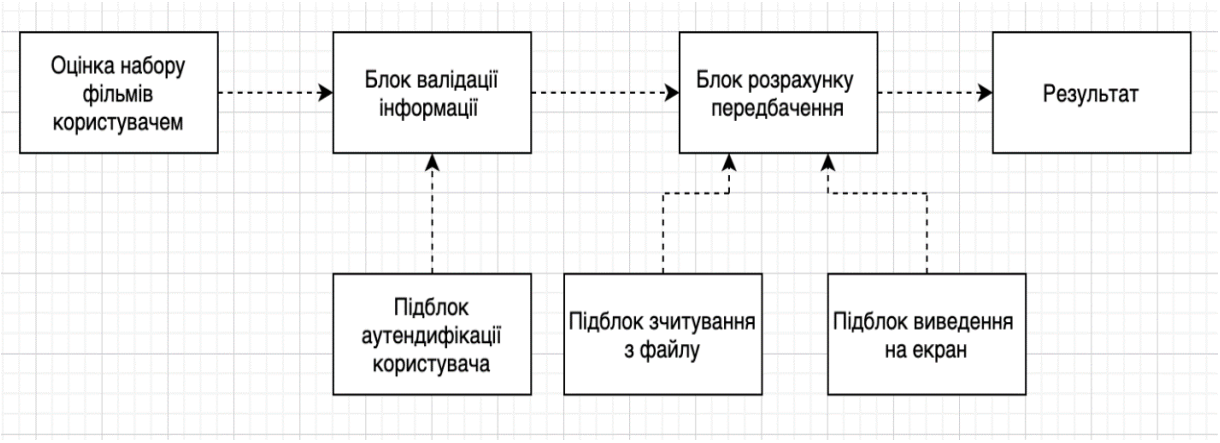

Рисунок 4.1 — Схема структури системи

4.1. Опис функціональності системи

Програмний застосунок для надання персоналізованих рекомендацій містить у собі одного головного актора – користувач системи.

На рисунку 4.2 представлена діаграма прецедентів, яка описує функції та дії актора у системі.

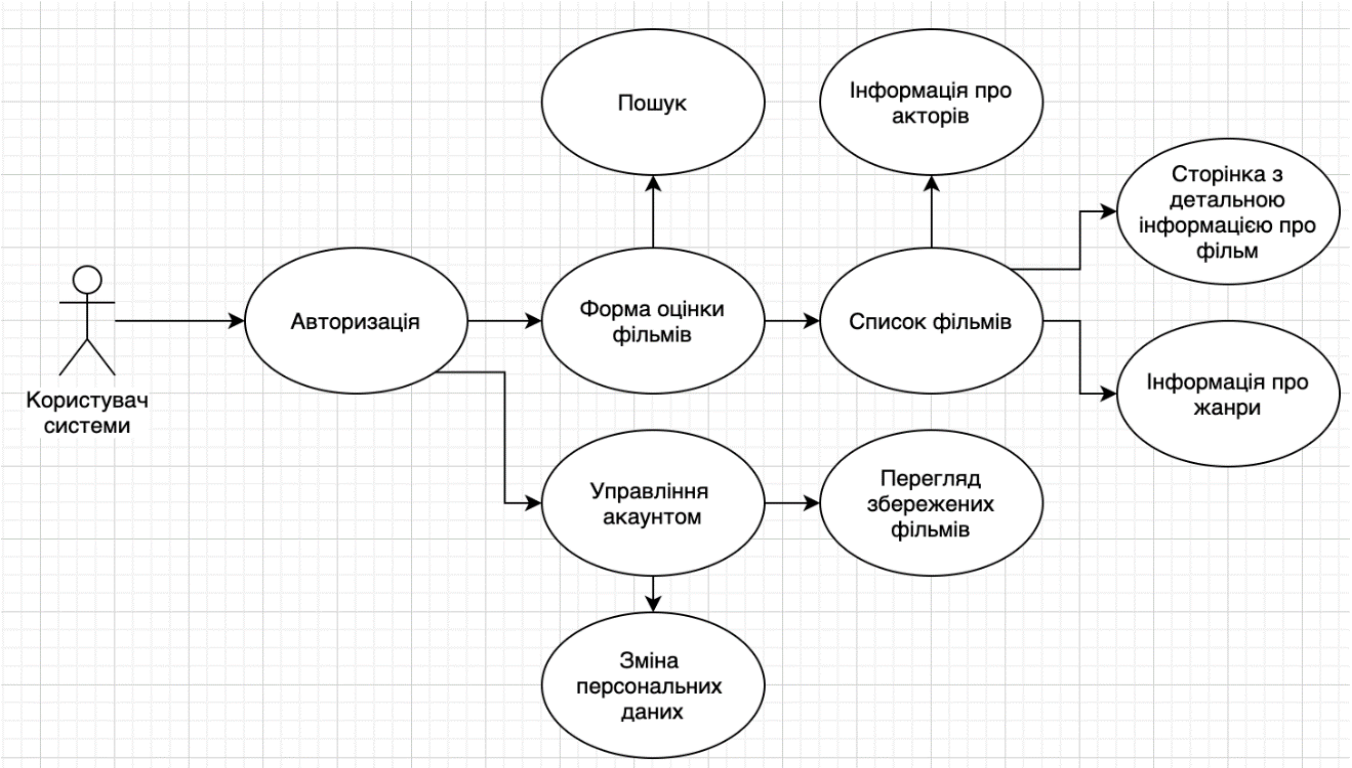

Рисунок 4.2 — Діаграма прецедентів системи

4.2. Концептуальна модель бази даних

База даних системи складається з п’яти взаємопов’язаних таблиць реляційної бази даних, які створюють єдиний інформаційний простір для зберігання та отримання доступу до даних.

Є дві основних таблиці: “Фільм” та “Користувач”. А також допоміжні таблиці такі як “Жанр” і “Фільм-Жанр”, остання створює зв’язок “багато-до-багатьох” між таблицями “Жанр” та “Фільм” . Таблиця “Рейтинг” містить інформацію про рейтинг, який користувач надав фільму.

Концептуальна модель бази даних приведена на рисунку 4.3.

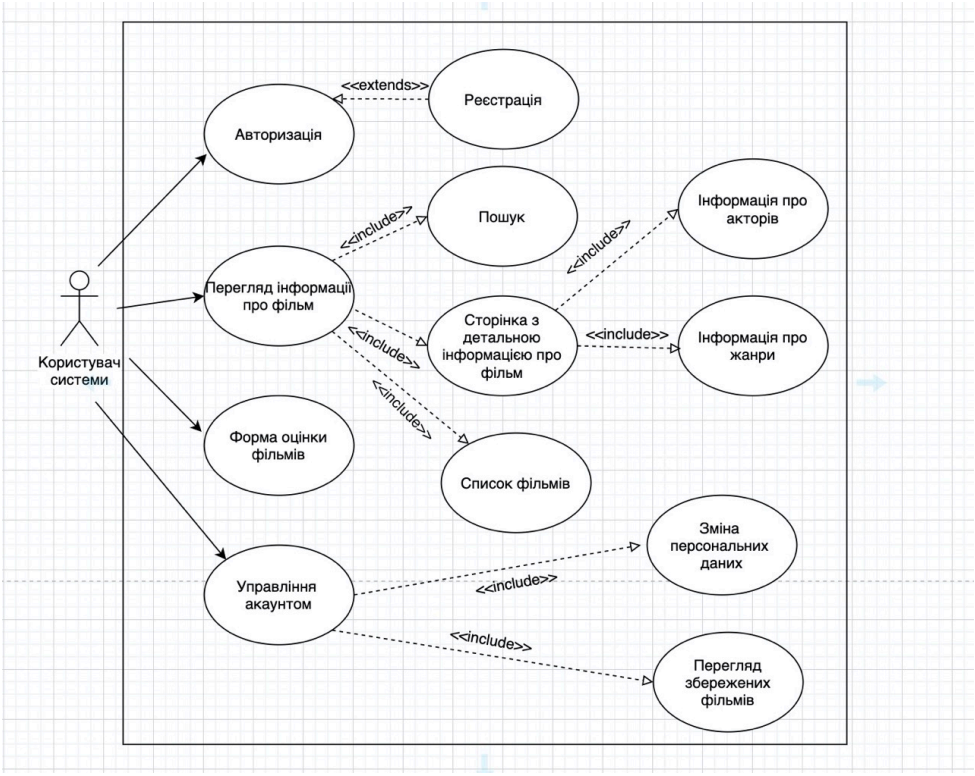

Рисунок 4.3 — Концептуальна модель БД

4.3. Опис таблиць бази даних

Для доступу до таблиць бази даних в програмі використовується Django ORM, що дає змогу зручно створювати з'єднання таблиць та отримувати дані з таблиць, які зв'язані як "багато-до-багатьох" чи "один-до багатьох".

Детальна інформація про їх структури (ім'я, тип і розмір поля, опис поля) приведена у таблицях 4.1 — 4.8.

Таблиця 4.1. Структура таблиці "Користувач"

Ім'я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
Username	nvarchar(100)	Логін користувача
Email	nvarchar(100)	Адреса електронної пошти
Password	nvarchar(10000)	Пароль
Avatar	blob	Зображення користувача
First_name	nvarchar(100)	Ім'я користувача
Second_name	nvarchar(100)	Фамілія користувача

Таблиця 4.2. Структура таблиці "Фільм"

Ім'я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
Title	nvarchar(100)	Назва фільму
Description	nvarchar(1000)	Опис фільму
Vote_average	decimal	Середня оцінка
Vote_count	integer	Загальна кількість оцінок

Таблиця 4.3. Структура таблиці “Фільм - Жанр”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
genre	nvarchar(11)	ID жанру
movie	nvarchar(11)	ID фільму

Таблиця 4.4. Структура таблиці “Жанр”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
Name	nvarchar(100)	Назва жанру
Descripition	nvarchar(1000)	Опис жанру

Таблиця 4.5. Структура таблиці “Рейтинг”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
User	nvarchar(11)	ID користувача
Movie	nvarchar(11)	ID фільму
Score	decimal	Рейтинг фільму

Таблиця 4.6. Структура таблиці “Персона”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
First name	nvarchar(255)	Ім’я
Last name	nvarchar(255)	Фамілія
Age	integer	Кількість років персоні

Таблиця 4.7. Структура таблиці “Роль”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
Type	nvarchar(11)	Назва ролі
Description	nvarchar(100)	Опис ролі

Таблиця 4.8. Структура таблиці “Персона-Роль ”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	nvarchar(11)	Первинний ключ
Person	nvarchar(11)	ID персони
Movie	nvarchar(11)	ID фільму
Role	nvarchar(11)	ID ролі

4.4. Розробка головної сторінки

Головна сторінка – це основне робоче місце користувача в системі. Вона є елементом компонування всієї системи

Підмодулі головної сторінки:

— модуль перегляду списку фільмів;

— модуль перегляд детальної інформації про фільм;

— модуль оцінки фільмів;

— модуль особистого кабінету користувача.

4.5. Модуль перегляду списку фільмів

Даний модуль дозволяє користувачу переглядати список фільмів та здійснювати операції з ними, а саме:

— Здійснювати пошук необхідного фільму;

— Зберігати фільм;

— Оцінювати фільм;

— Переходити до детальної інформації про фільм, кабінету користувача

— Фільтрувати фільми по жанрам

4.6. Модуль особистого кабінету користувача

Даний модуль дозволяє переглядати збережені фільми, а також змінювати персональну інформацію: пароль, пошту, зображення, ім’я.

4.7. Модуль оцінки фільмів

Даний модуль застосовується при першому вході користувача в систему. Він дозволяє оцінити користувачу запропоновані фільми та зберігає дану інформацію для надання персоналізованих рекомендацій.

4.8. Розробка алгоритму колаборативної фільтрації

Для запуску колаборативної фільтрації створена модель отримує дані від серверу з користувацькими вподобаннями. Як тільки користувач оцінив один з фільмів, модель активується і намагається надати передбачення (рисунок 4.4), змінюючи поточний порядок фільмів, який користувач бачить на екрані. Зробивши передбачення, потрібно знову тренувати модель з новими користувацькими даними. Це доволі затратний процес.

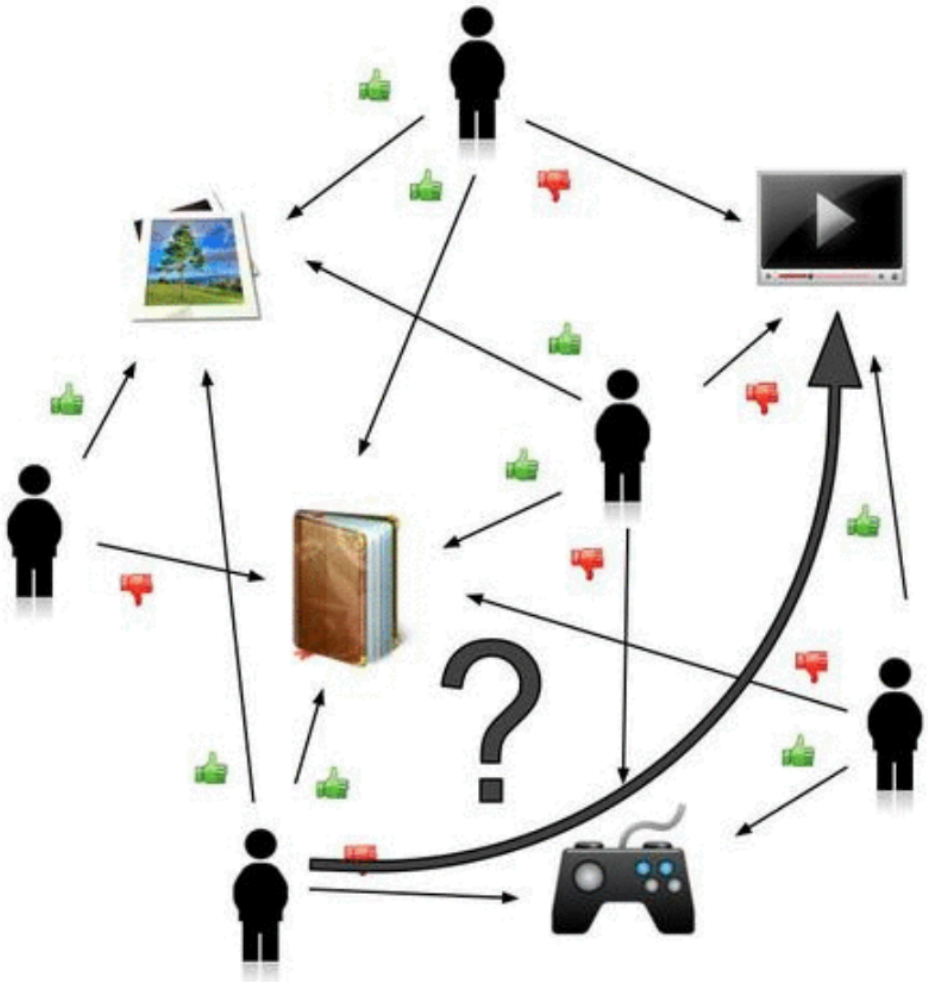

Рисунок 4.3. — Застосування колаборативної фільтрації

4.9. Розробка алгоритму надання персоналізованих рекомендацій на основі вмісту

Для визначення подібності на основі вмісту використовується перетворення опису фільму до вектору, приклад наведено на рисунку 4.4.

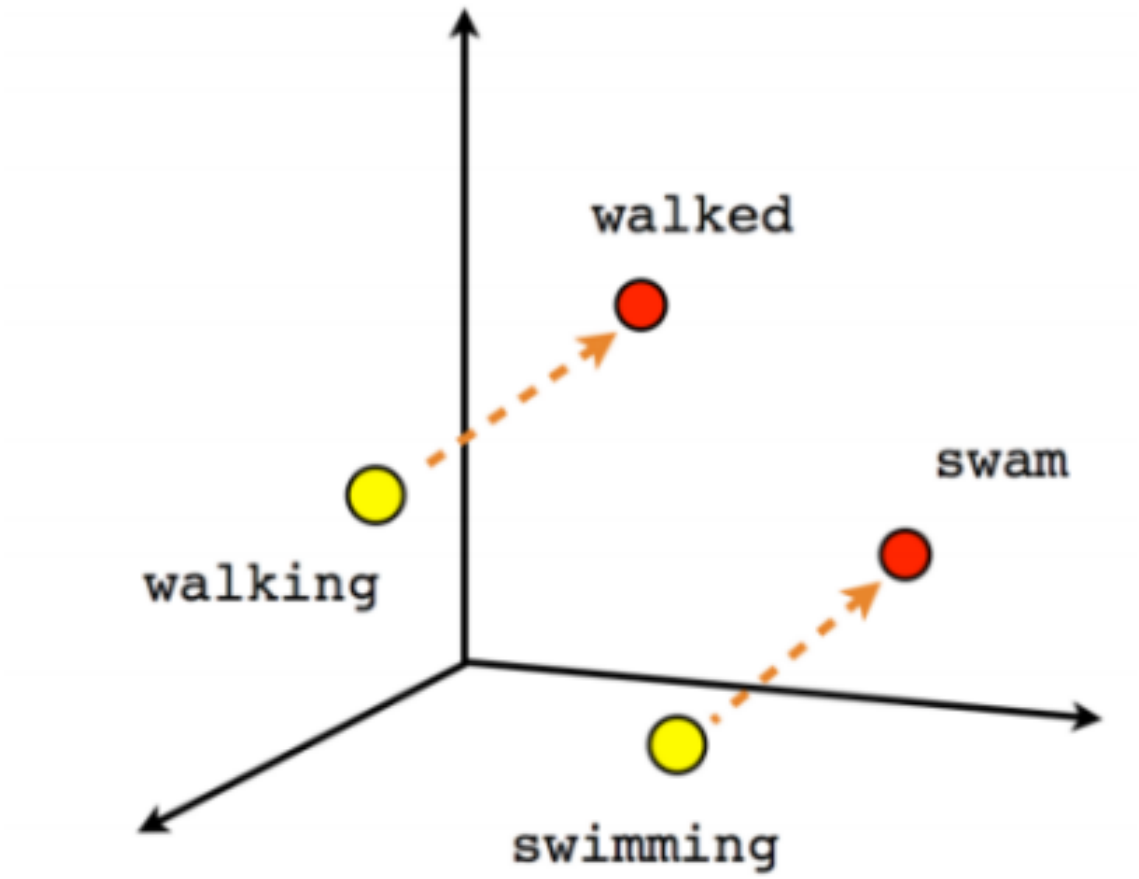

Рисунок 4.4. — Векторизація тексту

Перетворивши слова у вектори, складаємо матрицю подібності. В результаті отримуємо топ-10 найбільш подібних фільмів до даного.

Висновки до розділу 4

Таким чином, в результаті проектування та розробки рекомендаційної системи для надання персоналізованих рекомендацій, було створено веб-додатоок з мікросервісною архітектурою. Користувач може взаємодіяти з сервісом у мобільному телефоні чи за домомогою персонального комп’ютера у браузері, один з серверів відповідає за обробку даних від користувача, а інший за надання персоналізованих рекомендацій.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблена програмний комплекс розроблений з використанням веб-технологій і тому працює в браузерах, які підтримують актуальні веб-стандарти.

5.1. Інсталяція та системні вимоги

Для запуску даного веб-застосунку користувачу потрібен лише веб-браузер та підключення до мережі Інтернет. Оскільки розроблена система - це веб-додаток, користувач може скористатись ним не тільки з персонального комп’ютера, а й з мобільного телефону чи планшету.

5.2. Інструкція з використання програмного продукту

Додаток працює лише для авторизованих користувачів, тому спочатку потрібно зареєструватися в системі, це можна зробити за допомогою соціальних медіа. На рисунку 5.1 зображена форма авторизації.

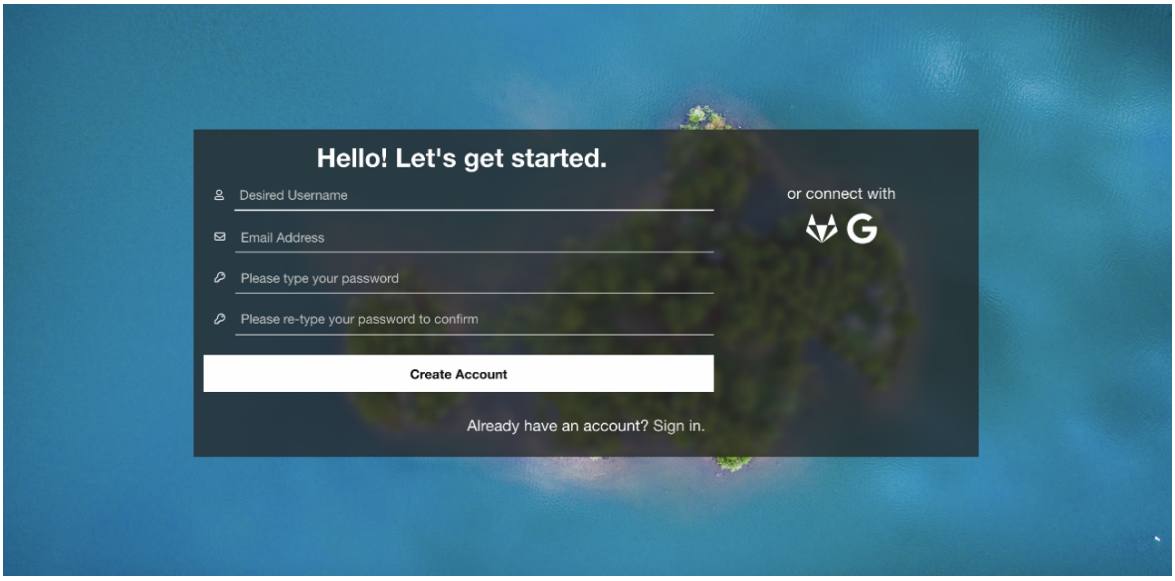

Рисунок 5.1 — Форма авторизації

Після того, як користувач увійшов до системи, додаток автоматично перенаправляє його до форми оцінки фільмів. Якщо користувач переглядав цей фільм, система просить оцінити його, якщо ні, то перейти до наступного кроку. На рисунку 5.2 зображена форма оцінки фільмів.

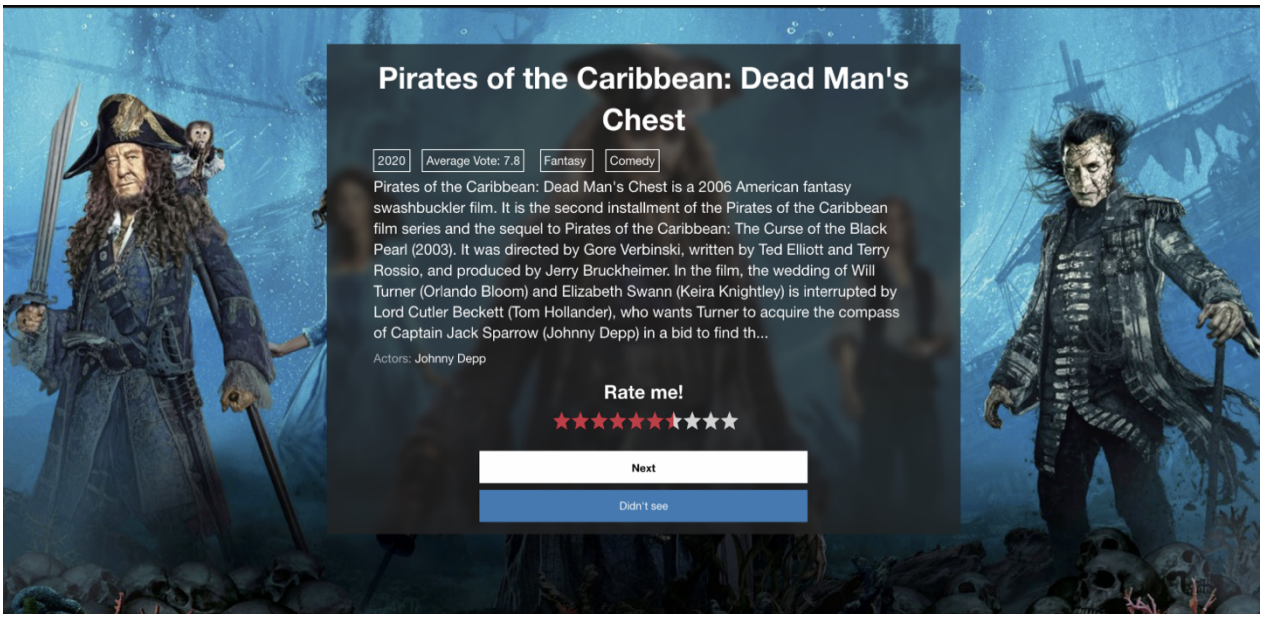

Рисунок 5.2 — Форма оцінки фільмів

Після успішного заповнення форми оцінки фільмів, система перенаправляє користувача до Головної сторінки, яка є основним компонентом системи.

Користувач може зберігати фільми, або переходити до сторінки детальної інформації про фільм чи до персонального кабінету. На рисунку 5.3 зображена головна сторінка додатку.

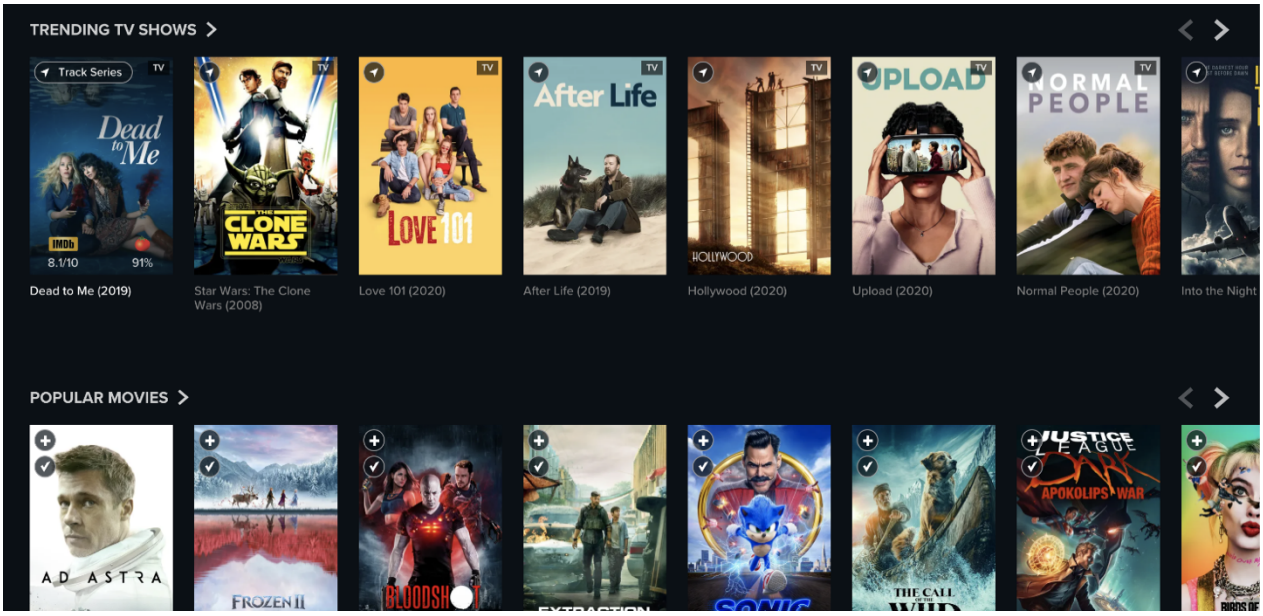

Рисунок 5.3 — Головна сторінка додатку

Користувач також може перейти в персональний кабінет, для зміни даних про нього. Він може змінити ім’я, пошту, пароль та інші параметри. На рисунку 5.4 зображений персональний кабінет користувача.

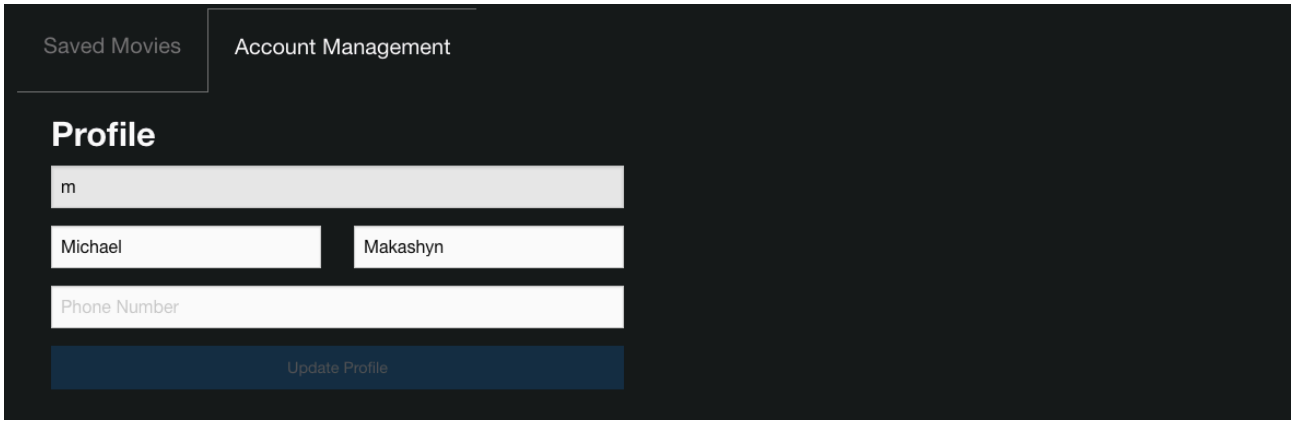

Рисунок 5.4 — Головне вікно керування камерою відеоспостереження

Однією з можливостей веб-додатку є збереження фільмів в особистому

кабінеті. Це дозволяє слідкувати за користувацькими вподобаннями і відповідно цього створювати передбачення (рисунок 5.5)

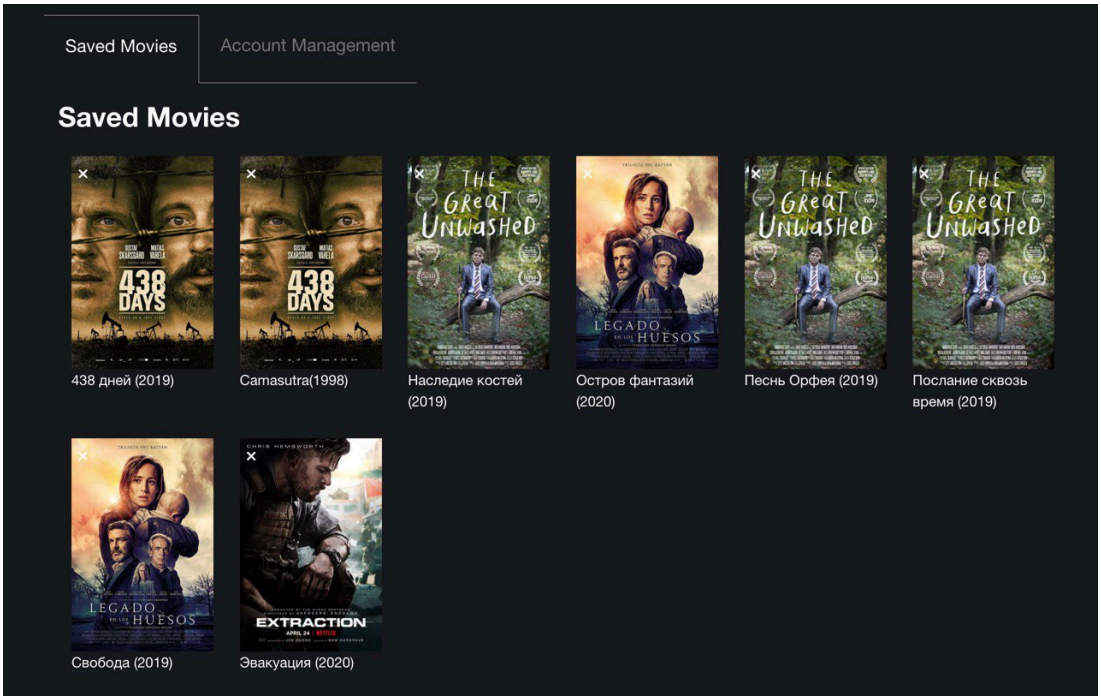

Рисунок 5.5 — Вікно збереження фільмі в особистому кабінеті користувача

Знаходячись на головній сторінці веб додатку, натиснувши на один із фільмів в каталозі, користувач може переглянути детальну інформацію про нього (Рисунок 5.6).

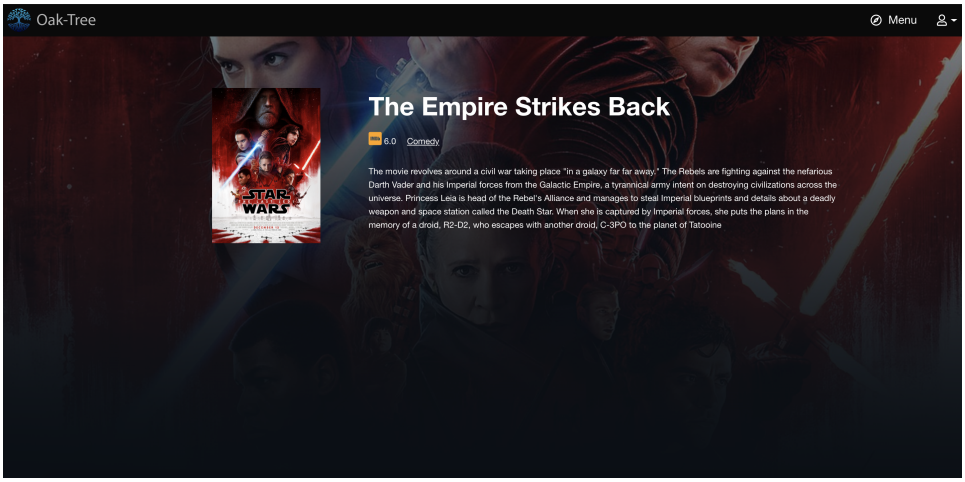

Рисунок 5.6 — Вікно перегляду детальної інформації про фільм

Висновки до розділу 5

Таким чином, в результаті було створено додаток для перегляду інформації про фільми, який демонструє реальне використання рекомендаційної системи для надання персоналізованих рекомендацій. Основними функціями цієї системи є авторизація та аутендифікація користувачів, зібрання даних про користувача та можливість перегляду інформації про фільми користувачем.

ВИСНОВКИ

У ході аналізу існуючих персоналізованих рекомендаційних систем було досліджено, що проблема ще не є вирішеною, адже складно передбачити можливі побажання користувача. Тому ще не було створено ідеальної рекомендаційної системи.

Побудований програмний продукт дозволяє надавати персоналізовані рекомендації фільмів користувачу, на основі його попередніх рішень, порівнюючи ці рішення з виборами інших користувачів. Для побудованої моделі були проведені тестуванні, які підтвердили працездатність додатку.

Були досліджені бібліотеки, фреймворки, методи та алгоритми машинного навчання, на основі яких будувалась і навчалась модель. Проведено порівняння видів рекомендаційних систем і вибрано найбільш ефективний спосіб для надання персональних рекомендацій. Були зроблені експерементні дослідження, з використанням різних алгоритмів машинного навчання: ALS, SVD, word2vec, tf-idf. Найефективніші з них були використані для тренування моделі.

Також було проведено дослідження, щодо використання існуючих веб-технологій, таких як Django, Backbone для розробки продукту і максимально зручної взаємодії з користувачем.

Отже, практика покращила знання в машинному навчанні та опрацюванні великих даних. Було досліджено різноманітні підходи до розробки додатків та покращено знання у роботі веб-додатків. В результаті цього було створено декілька алгоритмів машинного навчання, які лягли в основу програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алекс Шаф — Що таке веб додаток? [Електронний ресурс]. — 2008. — Режим доступу: http://www.jguru.com/faq/view.jsp?EID=129328.

2. Markus Egger — MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF [Електронний ресурс]. — 2012. — Режим доступу: https://www.packtpub.com/application-development/mvvm-survival-guide-enterprise-architectures-silverlight-and-wpf.

3. Martin Fowler — GUI Architectures. Часть 1 [Електронний ресурс]. — 2009. — Режим доступу: https://bit.ly/2CvCk1e.

4. Eric Matthes — Python Crash Course (2nd Edition) [Електронний ресурс]. — 2019. — Режим доступу: https://ebay.to/35YTKQC.

5. Больe А. — Learning SQL [Електронний ресурс]. — 2005. — Режим доступу: http://shop.oreilly.com/product/9780596007270.do.

6. Вес Маккіні — Python for Data Analysis: Data Wrangling with Pandas, Numpy [Електронний ресурс]. — 2011 — Режим доступу: https://amzn.to/3dNhREF.

7. Біл Чамберс. — Spark: The Def initive Guide: Big Data Processing Made Simple [Електронний ресурс]. — 2018. — Режим доступу: https://amzn.to/35Zwzpi

8. Surana, Ramit . Containerizing Docker on Kubernetes. [Електронний ресурс]. — 2015. – Режим доступу до ресурсу: https://bit.ly/3dE3XVW.

9. Bengio Y., Ducharme R., Vincent P. A neural probabilistic language model — 2003.

10. Jones K. S. A statistical interpretation of term specificity and its application in retrieval // Journal of Documentation : журнал. — MCB University : MCB University Press, 2004. — Т. 60, № 5. — С. 493-502.

11. Downey, Allen B. Think Python: How to Think Like a Computer Scientist — 2012

12. Zaharia, Matei; Chowdhury, Mosharaf; Das, Tathagata; Dave, Ankur; Ma, Justin; McCauley, Murphy; J., Michael; Shenker, Scott; Stoica,. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing — 2010

ДОДАТОК 1

Рекомендаційна система з використанням методів машинного навчання для надання персоналізованих рекомендацій в реальному часі

Текст програми

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТМ

Аркушів 10

Київ – 2020

```
import os, posixpath, socket
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

# General Settings
SPARK_EXECIMAGE = 'code.oak-tree.tech:5005/courseware/oak-tree/dataops-examples/spark245-k8s-minio-ml'
SPARK_EXECUTORS = 4

# Set python versions explicitly using the PySpark environment variables
# to prevent the executors from using the wrong version of Pytho9
os.environ['PYSPARK_PYTHON'] = 'python3'
os.environ['PYSPARK_DRIVER_PYTHON'] = 'python3'

# Spark Configuration
conf = SparkConf()

# Set Spark Master to the local Kubernetes Driver
conf.setMaster('k8s://https://kubernetes.default.svc')

# Configure executor image and runtime options
conf.set('spark.kubernetes.container.image', SPARK_EXECIMAGE)
conf.set("spark.kubernetes.authenticate.driver.serviceAccountName", K8S_SERVICEACCOUNT)
conf.set("spark.kubernetes.namespace", K8S_NAMESPACE)

# Spark on K8s works ONLY in client mode (driver runs on client)
conf.set('spark.submit.deployMode', 'client')
conf.set("spark.driver.port", "20020")

# Executor instances and settings
conf.set('spark.executor.instances', SPARK_EXECUTORS)

# Configure S3 Object Storage as filesystem, pass MinIO credentials
conf.set("spark.hadoop.fs.s3a.endpoint", 'https://storage.centerville.oak-tree.tech') \
    .set("spark.hadoop.fs.s3a.access.key", 'oaktree') \
    .set("spark.hadoop.fs.s3a.secret.key", 'object-storage@oak-tree.office') \
    .set("spark.hadoop.fs.s3a.fast.upload", True) \
    .set("spark.hadoop.fs.s3a.path.style.access", True) \
```

```
    .set("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")

# Application Name
conf.setAppName('movie-rival-train')

# Set IP of driver. This is always the user pod. The socket methods
# dynamically fetch the IP address.
conf.set('spark.driver.host', socket.gethostbyname(socket.gethostname()))
conf.set('spark.driver.memory', '16g')

# Initialize the SparkContext
sc = SparkContext(conf=conf)
spark = SparkSession(sc)
spark.version

movies_df = spark.read.format('csv').option('header', True).load('s3a://jupyter.oak-tree.tech/movie-
rival/movies.csv')
movies_df.take(1)

ratings_df = spark.read.format('csv').option('header', True).load('s3a://jupyter.oak-tree.tech/movie-
rival/ratings.csv')
ratings_df.take(1)

movies_rdd = movies_df.rdd.map(tuple)
movies_rdd.take(1)

movies_filtered = movies_rdd.map(lambda r: (int(r[0]), r[1][:-6].strip(), r[1][-
6:].replace('(','').replace(')','').replace('"','')) )
movies_filtered.take(10)

def tf_try(v):
    try:
        int(v)
        #f(*args, **kwargs)
        return True
    except:
        return False
```

```
test = movies_filtered.filter(lambda r: tf_try( r[2] ))
test.take(5)

movies_normalized = test.map(lambda r: (int(r[0]), r[1], int(r[2])) )
movies_normalized.take(5)

# Filter out all movies that are older than 2010
recent_movies = movies_normalized.filter(lambda r: r[2] >= 2)
recent_movies = sc.parallelize(recent_movies.take(5))

ratings_rdd = ratings_df.rdd.map(tuple)
ratings_rdd.take(5)

ratings_normalized = ratings_rdd.map(lambda r: ( int(r[0]), int(r[1]), float(r[2]) ))
ratings_normalized.take(5)

all_movie_ids = recent_movies.map(lambda r: r[0]).collect()
all_movie_ids[:10]

ratings_recent = ratings_normalized.filter(lambda r: r[1] in all_movie_ids )
ratings_recent.take(5)

df = spark.createDataFrame(ratings_recent).toDF("user_id", "movie_id", "rating")
df.take(5)

(training, test) = df.randomSplit([0.8, 0.2])

from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics

als = ALS(maxIter=10, rank=8, regParam=0.1, userCol="user_id", itemCol="movie_id", ratingCol="rating",
        coldStartStrategy="drop")
model = als.fit(training)
```

```
# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

### Content based recommendation
movies_df.take(5)
movies_filtered = movies_rdd.map(lambda r: (int(r[0]),
                                            r[1][:-6].strip(), r[1][-6:].replace('(','').replace(')','').replace('',''),
                                            str(r[2]).replace('|', ', ')),)
movies_filtered.take(5)
movies_df = spark.createDataFrame(movies_filtered).toDF("movie_id", "title", "released", "genre").toPandas()
movies_df.head()

# we need more movie data here, keywords, probably description
# wilil use title and genres for now
from rake_nltk import Rake
from sklearn.feature_extraction.text import CountVectorizer

movies_df['Key_words'] = ''

for index, row in movies_df.iterrows():
    plot = row['genre']

    # instantiating Rake, by default it uses english stopwords from NLTK
    # and discards all puntuation characters as well
    r = Rake()

    # extracting the words by passing the text
    r.extract_keywords_from_text(plot)

     # getting the dictionary whith key words as keys and their scores as values
    key_words_dict_scores = r.get_word_degrees()

    # assigning the key words to the new column for the corresponding movie
    row['Key_words'] = list(key_words_dict_scores.keys())
```

```
# instantiating and generating the count matrix
count = CountVectorizer()
count_matrix = count.fit_transform(movies_df['genre'])

# generating the cosine similarity matrix
cosine_sim = cosine_similarity(count_matrix, count_matrix)

# creating a Series for the movie titles so they are associated to an ordered numerical
# list I will use in the function to match the indexes
indices = pd.Series(df.index)

#  defining the function that takes in movie title
# as input and returns the top 10 recommended movies
def recommendations(title, cosine_sim = cosine_sim):

    # initializing the empty list of recommended movies
    recommended_movies = []

    # gettin the index of the movie that matches the title
    idx = indices[indices == title].index[0]

    # creating a Series with the similarity scores in descending order
    score_series = pd.Series(cosine_sim[idx]).sort_values(ascending = False)

    # getting the indexes of the 10 most similar movies
    top_10_indexes = list(score_series.iloc[1:11].index)

    # populating the list with the titles of the best 10 matching movies
    for i in top_10_indexes:
        recommended_movies.append(list(df.index)[i])

    return recommended_movies



import logging, posixpath

from django.db import models
from django.core.exceptions import ImproperlyConfigured
```

```
from django.shortcuts import reverse
from django.contrib.auth import get_user_model

from wagtail.core.models import index
from wagtail.admin.edit_handlers import FieldPanel, FieldRowPanel, MultiFieldPanel, InlinePanel
from wagtail.images.models import AbstractImage, AbstractRendition
from wagtail.core.models import Orderable
from modelcluster.fields import ParentalManyToManyField, ParentalKey
from modelcluster.models import ClusterableModel

from guru.models import GuruTokenModel
from guru.helpers.utils.object import pick
from guru import apisettings as gapicodes

from .genres import Genre
from .category import Category
from .members import Person, Role

from ..edit_handlers import GenreChooserPanel, PersonChooserPanel, CategoryChooserPanel

logger = logging.getLogger(__name__)


# Movie status choices
MOVIE_STATUS_RELEASED = 'RED'
MOVIE_STATUS_RUMORED = 'RUD'
MOVIE_STATUS_POST_PRODUCTION = 'PPR'
MOVIE_STATUS_IN_PRODUCTION = 'IPR'
MOVIE_STATUS_PLANNED = 'PLD'
MOVIE_STATUS_CANCELED = 'CLD'


MOVIE_STATUSES = (
    (MOVIE_STATUS_RELEASED, 'Released'),
    (MOVIE_STATUS_RUMORED, 'Rumored'),
    (MOVIE_STATUS_POST_PRODUCTION, 'Post Production'),
    (MOVIE_STATUS_IN_PRODUCTION, 'In Production'),
    (MOVIE_STATUS_PLANNED, 'Planned'),
```

```
    (MOVIE_STATUS_CANCELED, 'Canceled')
)

MOVIE_STATUSES_LOOKUP = dict(MOVIE_STATUSES)

#  Movie original language choices
MOVIE_LANGUAGE_ENGLISH = 'EN'
MOVIE_LANGUAGE_GERMAN = 'DE'
MOVIE_LANGUAGE_CHINESE = 'ZH'
MOVIE_LANGUAGE_FRENCH = 'FR'
MOVIE_LANGUAGE_ITALIAN = 'IT'
MOVIE_LANGUAGE_PERSIAN = 'FA'
MOVIE_LANGUAGE_DUTCH = 'NL'


MOVIE_LANGUAGES = (
    (MOVIE_LANGUAGE_ENGLISH, 'English'),
    (MOVIE_LANGUAGE_GERMAN, 'German'),
    (MOVIE_LANGUAGE_CHINESE, 'Chinese'),
    (MOVIE_LANGUAGE_FRENCH, 'French'),
    (MOVIE_LANGUAGE_ITALIAN, 'Italian'),
    (MOVIE_LANGUAGE_PERSIAN, 'Persian'),
    (MOVIE_LANGUAGE_DUTCH, 'Dutch')
)

MOVIE_LANGUAGES_LOOKUP = dict(MOVIE_LANGUAGES)


class Movie(index.Indexed, GuruTokenModel, ClusterableModel):
    ''' Describes Movie table
    '''
    title = models.CharField(max_length=255, help_text='Title of the movie')
    status = models.CharField(max_length=3, choices=MOVIE_STATUSES,
default=MOVIE_STATUS_RELEASED,
        help_text='Current movie status (Released, Rumored, Post Production, In Production, Planned, Canceled)')
    budget = models.DecimalField(decimal_places=2, max_digits=20, null=True, blank=True,
        help_text='Movie budget')
    revenue = models.DecimalField(decimal_places=2, max_digits=20, null=True, blank=True,
```

```
        help_text='Movie revenue')

    adult = models.BooleanField(default=False, help_text='Movie age restrictions')

    imdb_id = models.CharField(null=True, blank=True, max_length=9,
        help_text='Movie id in the IMDB')
    original_language = models.CharField(null=True, blank=True, max_length=2,
choices=MOVIE_LANGUAGES,
        help_text='Movie original language')

    overview = models.TextField(help_text='Movie overview')

    release_date = models.DateField(null=True, blank=True,
        help_text='Movie release date')
    runtime = models.DecimalField(max_digits=5, decimal_places=1, null=True, blank=True,
        help_text='Movie runtime')
    tagline = models.CharField(max_length=1000, null=True, blank=True,
        help_text='Movie tagline (slogan)')
    vote_average = models.DecimalField(max_digits=3, decimal_places=1, null=True, blank=True,
        help_text='The vote average for movie')
    vote_count = models.IntegerField(null=True, blank=True,
        help_text='Total count of votes for movie')
    ctime = models.DateTimeField(auto_now_add=True, editable=False, verbose_name='Created')
    mtime = models.DateTimeField(auto_now=True, editable=False, verbose_name='Modified')


    class Meta:
        app_label = 'movierival'
        verbose_name = 'Movie'
        verbose_name_plural = 'Movies'
        unique_together = ('title', 'release_date')
        ordering = ('title', 'vote_average')


    search_fields = [
        # Search fields
        index.SearchField('title'),
        index.SearchField('status'),
```

```
    index.SearchField('imdb_id'),

    index.SearchField('overview'),
    index.FilterField('runtime'),
    index.FilterField('vote_average'),
    index.FilterField('vote_count'),
    index.FilterField('original_language'),
    index.FilterField('adult'),
    index.FilterField('budget'),
    index.FilterField('revenue'),

    index.RelatedFields('genres', [
        index.SearchField('name'),
    ]),

    index.RelatedFields('cast_and_crew', [
        index.RelatedField('person', [
            index.SearchField('first_name'),
            index.SearchField('last_name'),
            index.FilterField('age')
        ]),
        index.RelatedField('role', [
            index.SearchField('name'),
            index.SearchField('description')
        ])
    ])
]


panels = [
    FieldPanel('title'),
    FieldPanel('overview'),
    FieldPanel('tagline'),
    InlinePanel('genres', label='Genre'),
    InlinePanel('category', label='Category'),
    InlinePanel('cast_and_crew', label='Cast and Crew'),
    MultiFieldPanel([
        FieldRowPanel([
```

```
            FieldPanel('status'),
            FieldPanel('original_language'),
            FieldPanel('release_date')
        ]),
        FieldRowPanel([
            FieldPanel('budget'),
            FieldPanel('revenue'),
        ]),
        FieldRowPanel([
            FieldPanel('adult'),
            FieldPanel('runtime')
        ]),
        FieldRowPanel([
            FieldPanel('vote_average'),
            FieldPanel('vote_count'),
            FieldPanel('imdb_id'),
        ])
    ], 'Movie Details')
]

def __str__(self):
    return 'Movie: %s' % self.title

@property
def json(self):

    mdata = {
        'movie_score_url': reverse('movierival:api:movie-score'),
        'cast_and_crew': self.cast_and_crew_json,
        'genres': self.genres_json[0] if isinstance(self.genres_json, tuple) else self.genres_json,
        'movie_detail_url': reverse('movierival:movie-detail', args=[self.pk]),
        'saved': self.saveCheck
    }
    # Core properties
    mdata.update(pick(self, [f.name for f in self._meta.fields]))

    return mdata
```

ДОДАТОК 2

Рекомендаційна система з використанням методів машинного навчання для надання персоналізованих рекомендацій в реальному часі

Опис програми

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТМ

Аркушів 9

Київ – 2020

АНОТАЦІЯ

Додаток надає персоналізовані рекомендації користувачу. Створений програмний продукт дає можливість:

— визначати фільм, який з найбільшої вірогідністю сподобається користувачу на основі попередніх прийнятих рішень іншими користувачами.
— визначати подібності фільмів за його описом.
— отримувати персоналізовані рекомендацій користувачу.
— переглядати список фільмів, детальну інформацію про фільм та акторів
— реєстрації та авторизації нового користувача
— оцінювати та зберігати фільми до особистого кабінету користувача

Дана система розроблювалась за допомогою мови програмування Python та JavaScript. Для розробки використовувались фреймворк Django та Backbone.js. Для реалізації моделі використовувались технології Apache Spark, Kubernetes, Kafka. База даних — Postgresql.

ЗМІСТ

-4-

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до назви дипломної роботи, розроблений продукт надає персоналізовані рекомендації користувачу в реальному часі.

Користувач, для роботи з системою, повинен мати комп’ютер або мобільний телефон.

Для використання додатку потрібно запустити веб браузер та перейти за посиланням.

-5-

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний засіб надає персоналізовані рекомендації користувачу в реальному часі, на основі попередніх дій користувача у веб-застосунку. Для цього було створено колаборативну рекомендаційну систему, а також рекомендаційну систему на основі вмісту. Програмний продукт дозволяє взаємодіяти з рекомендаційними системами та показувати користувачу той контент, який йому з найбільшою ймовірністю сподобається.

-6-

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Загальний принцип роботи додатку:

1) Користувач заходить на веб-сайт та може переглядати список фільмів, детальну інформацію про них, здійснювати пошук по веб сайті.
2) Для того, щоб оцінювати чи зберігати фільми користувач повинен пройти авторизацію або реєстрацію.
3) Після проходження авторизації на сайті користувач може оцінювати фільми і зберігати їх до особистого кабінету.
4) Після цього система отримує дані від користувача та відповідно їх впорядковує фільми у веб-додатку відповідно до вподобань користувача.
5) Користувач може керувати особистим кабінетом, переглядати збережені фільми в ньому, а також змінювати інформацію про себе.

-7-

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Дана система розроблювалась за допомогою мови програмування Python та JavaScript. Для розробки використовувались фреймворк Django та Backbone.js. Для реалізації моделі використовувались технології Apache Spark, Kubernetes, Kafka. База даних — Postgresql.

-8-

ВХІДНІ І ВИХІДНІ ДАНІ

Користувач оцінює фільми, надаючи наступні дані:

— Рейтинг, який користувач присвоїв фільму

— ID фільму

Вихідними даними є:

— Список фільмів впорядкований за вподобаннями користувача, а також на основі подібності опису, акторів, жанрів.